NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

PROBABILISTIC SEARCH ON OPTIMIZED GRAPH TOPOLOGIES

by

Christian Klaus

September 2011

Thesis Co-Advisors:	Timothy H. Chung
	Craig Rasmussen
Second Readers:	Nedialko Dimitrov
	Ralucca Gera

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* 23–09–2011	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* 2009-07-01—2011-09-31
4. TITLE AND SUBTITLE Probabilistic Search on Optimized Graph Topologies		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Christian Klaus		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Department of the Navy		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. **IRB Protocol Number: N.A.**

14. ABSTRACT

This thesis investigates how the performance of a mobile searcher is affected by altering the search environment. We model the search environment as a simple connected, undirected graph. By adding new edges to the graph, we change the search environment. Our objective is to optimize search performance, that is, to minimize the (expected) time needed to find the target, in the context of probabilistic search. We first analyze two different methods to generate random connected graphs, then evaluate a number of methods to augment the graph, typically by considering the algebraic connectivity of the graph and its associated (Fiedler) eigenvector. Extensive simulation studies and resulting statistical and theoretical models show that adding a few wisely chosen edges to a sparse graph is sufficient to dramatically increase search performance. Further, we propose a novel method for incorporating prior information about the target's likely location by defining a subgraph on which the presented approach is performed, resulting in even greater improvements in search performance.

15. SUBJECT TERMS

search and detection, search environment, graph Laplacian, algebraic connectivity, augmenting a graph

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(include area code)*
Unclassified	Unclassified	Unclassified	UU	81	

THIS PAGE INTENTIONALLY LEFT BLANK

PROBABILISTIC SEARCH ON OPTIMIZED GRAPH TOPOLOGIES

Christian Klaus
Major, German Army
Diplom-Ingenier in Information Engineering

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

and

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
September 2011

Author:	Christian Klaus	
Approved by:	Timothy H. Chung Thesis Co-Advisor	Craig Rasmussen Thesis Co-Advisor
	Nedialko Dimitrov Second Reader	Ralucca Gera Second Reader
	Robert F. Dell Chair, Department of Operations Research	
	Carlos Borges Chair, Department of Applied Mathematics	

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis investigates how the performance of a mobile searcher is affected by altering the search environment. We model the search environment as a simple connected, undirected graph. By adding new edges to the graph, we change the search environment. Our objective is to optimize search performance, that is, to minimize the (expected) time needed to find the target, in the context of probabilistic search. We first analyze two different methods to generate random connected graphs, then evaluate a number of methods to augment the graph, typically by considering the algebraic connectivity of the graph and its associated (Fiedler) eigenvector. Extensive simulation studies and resulting statistical and theoretical models show that adding a few wisely chosen edges to a sparse graph is sufficient to dramatically increase search performance. Further, we propose a novel method for incorporating prior information about the target's likely location by defining a subgraph on which the presented approach is performed, resulting in even greater improvements in search performance.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 INTRODUCTION **1**
1.1 Background . . . 1
1.2 Objectives . . . 2
1.3 Literature Review . . . 3
1.4 Scope . . . 5
1.5 Main Contributions . . . 5
1.6 Organization of Thesis . . . 5

2 MATHEMATICAL FORMULATION **7**
2.1 Problem Formulation . . . 7
2.2 Probabilistic Search Review . . . 7
2.2.1 Definitions and Notations . . . 8
2.2.2 Some Search Behaviors . . . 11
2.3 Graph-Theory Review . . . 12
2.3.1 Definitions and Notations . . . 12
2.3.2 The Graph Laplacian . . . 13
2.3.3 On Random Walks . . . 14
2.4 Optimization Review . . . 15
2.4.1 Brute Force . . . 15
2.4.2 Fiedler Vector Method . . . 15
2.4.3 Semidefinite Program (SDP) . . . 16

3 SIMULATION RESULTS AND ANALYSIS **21**
3.1 Generating Random Simple, Connected Graphs . . . 21
3.1.1 The Spanning-Tree Method . . . 21
3.1.2 The Acceptance-Rejection Method . . . 22
3.1.3 Comparing the Methods by Maximum Degree . . . 22
3.2 Sequential vs. Simultaneous Adding of Edges . . . 23
3.3 Fiedler Vector vs. SDP . . . 26
3.4 Relationship between TTD and Algebraic Connectivity . . . 28

3.5 Growing Graphs by the Greedy Algorithm 30
3.6 Incorporating Initial Information 37
3.6.1 The Subgraph Idea. 37
3.6.2 Coordinates and a More Realistic Edge Length 39
3.7 Other Search Methods 43
3.7.1 The Curse of a Myopic Searcher with Perfect Sensor 44
3.7.2 Shortest-path Searcher 45
3.8 Growing Graphs by Hitting Time 46

4 Conclusion and Future Work **49**
4.1 Conclusion. 49
4.2 Future Work 50

Appendices **59**

A Software Used for Simulation and Analysis **59**

Initial Distribution List **61**

List of Figures

Figure 2.1 Initial probability map with belief one 9

Figure 2.2 Initial probability map with belief one half 9

Figure 2.3 Belief of target presence 11

Figure 3.1 Comparison of the two random graph-generating methods 23

Figure 3.2 Ten-node graph, sequential adding of first edge 25

Figure 3.3 Ten-node graph, sequential adding of second edge 25

Figure 3.4 Ten-node graph, simultaneous adding of two edges 25

Figure 3.5 Ten-node graph, simultaneous adding of three edges 25

Figure 3.6 Fiedler vs. SDP, add one edge to a random graph 26

Figure 3.7 Fiedler vs. SDP, add more than one edge to a random graph 27

Figure 3.8 Different methods to grow a graph 28

Figure 3.9 Connectivity in relation to number of edges 29

Figure 3.10 TTD in relation to graph connectivity 29

Figure 3.11 Change of TTD with adding edges by greedy algorithm 30

Figure 3.12 Conducting searches on fifteen growing sparse graphs with 20 nodes . 32

Figure 3.13 Conducting searches on fifteen growing sparse graphs with 50 nodes . 33

Figure 3.14 Conducting searches on ten growing sparse graphs with 100 nodes . . 33

Figure 3.15 Estimate coefficient a 35

Figure 3.16 Estimate coefficient b 35

Figure 3.17 Estimate coefficient c 36

Figure 3.18 Induced subgraph 38

Figure 3.19 Subgraph analysis, random graphs 39

Figure 3.20 Graph with assigned coordinates and prior target distribution 40

Figure 3.21 TTD conditioned on target location 42

Figure 3.22 Subgraph analysis, random graphs 43

Figure 3.23 Example for a perfect searchers trap 44

Figure 3.24 Hitting-time method vs. Fiedler method 48

List of Tables

Table 3.1 TTD for different start nodes 44

Table 3.2 Statistics for a trapped perfect myopic sensor 45

THIS PAGE INTENTIONALLY LEFT BLANK

Executive Summary

High quality information gathered in a short time leads to good decision making. Search and detection systems are designed to provide information. The effectiveness of a search and detection system is dependent on the search algorithms used, the accuracy of detection sensors, and the search environment. This thesis is focused on altering the search environment to increase search performance, that is, to decrease the expected time it takes a single searcher to find an object, the target, inside the search environment.

We model the search environment as a simple connected, undirected graph. The nodes of the graph represent the discrete search cells. An edge between two nodes exists if the search cells are accessible from each other in the search environment. A searcher is restricted to moving along edges of the graph as he searches the environment.

The target's location is fixed and randomly chosen according to a probability map showing the likelihood of the target's presence at each search cell. Due to the lack of prior intelligence, the target could be located to any cell equal likely and the probability map is non-informative, i.e, uniform. Incorporating prior knowledge about the target's location requires reasonable probability models to calculate the probability map. A Markov chain is handy to do those calculations. Assuming the target's exact location at time zero, the Markov chain calculation provides probability distributions for any time late, that is, the delay time until the search begins.

Our objective is to find the edges we have to add to the graph in order to maximize the search performance. It is often computationally intractable to analytically compute the expected time to find the target; however, we can always estimate the quantity through simulation. While simulation can be used to evaluate the goodness of the added edge, it does not help to optimize the edge selecting process. We do not want to add edges randomly in a brute force kind of manner and evaluate our choice by simulation. We prefer to add edges in an optimal manner. Therefore we need a measurement for goodness that allows us to put the edges in order.

The algebraic connectivity — the second-smallest eigenvalue of the graph Laplacian — has been shown to serve as a good measure to judge the goodness of an added edge. We find a heuristic relationship between algebraic connectivity and time to find the target. The algebraic connectivity is non-deceasing with adding edges to the graph. Increasing algebraic connectivity will most likely decrease the time it takes to find the target. Besides brute force, we could neither find nor develop an exact method that meets our desire of finding the optimal set of edges to add. However, we demonstrate that a heuristic, greedy algorithm that utilizes the information of the Fiedler vector — the eigenvector associated to the second-smallest eigenvalue of the graph Laplacian — is a very effective and efficient method to maximize the algebraic connectivity. We compare this method to an algorithm that uses a semidefinite program to maximize the algebraic connectivity by adding edges. Using the greedy algorithm is supported by the analysis of the resulting search times.

In addition to increasing the algebraic connectivity of the graph, the maximum hitting time can be used to grow a graph as well. A uniform probability map does not provide information to the searcher about the next move. The searcher will start a random walk through the graph. The hitting time is the time it takes a randomly walking searcher to travel between any two nodes of the graph. Large hitting times in a graph cause large search times. Motivated by this fact, we place an edge between the nodes that have the maximum hitting time.

Our simulations have shown that it is sufficient to add a few edges to the graph chosen by the greedy algorithm. The gain in search performance incurred by adding edges vanishes with graph density. The decrease in time to find the target follows an exponential trend with augmenting the graph and approaches the expected time it takes to find the target in a complete graph. We propose approximation formulas to estimate the parameters for the exponential trend line.

By constraining the growing algorithm to consider only a special subset of the nodes, we can increase search performance even further. Intuitively, we would like the searcher to search the cells with a higher probability of the target's presence with priority. We extract an induced subgraph containing only cells with a high likelihood of finding the target. To determine the size of the subgraph, we use the entropy of the probability map as a measure. The subgraph is subjected to augmentation by the proposed methods before it is merged back into the original graph.

Simulation studies show that the results hold whether we use a myopic searcher or a searcher that finds its way to the next desired search cell using a shortest path algorithm. The main part of the simulation was done utilizing a perfect myopic searcher. Although the absence of false detection rates may appear to oversimplify the search problem, it shows interactions with the environment. The searcher could be trapped just because of the nature of the myopic behavior. Using a shortest path searcher supports the proposed methods to augment the graph in order to increase search performance.

As a side product, the reader will find comparing analysis of two algorithms to generate random connected graphs. Looking for a way to generate random graphs that can be used as a search environment, we find that a time-consuming acceptance–rejection method can be used to ensure the graph is connected.

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgements

First and foremost, I would like to thank Professor Timothy H. Chung, my thesis advisor, and Professor Nedialko Dimitrov. You have opened a world of knowledge and interest to me beyond the limits of this thesis study through your guidance, patience, and many open-minded discussions.

In addition, I am thankful to my co-advisor, Professor Craig Rasmussen, and Professor Ralucca Gera from the Department of Applied Mathematics. I was highly encouraged by your passion in mathematics to pursue a dual master's degree.

Lastly, but certainly not least, I would like to thank my wife, Ilka, and our outstanding sons, Julius and Eric Ethan. Certainly I would not be where I am today if it were not for your love and support. I am truly grateful and blessed by you each and every day.

In memory of Major Thomas Tholi, German Army,
who died in Afghanistan on May 28th, 2011.
You were an honorable officer, truly a comrade and friend.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
INTRODUCTION

1.1 Background

The three main parts in ground battle, such as in urban and irregular warfare, are fire, movement ability, and resistance against adversary weapon force. These military skills have determined the strength of an armed force for ages. As a fourth battle factor, information dominance has recently been added due to the invention and dramatic improvement of computer systems in the last half century.

Knowledge about the adversary's state can be a huge advantage and can influence an operation's outcome. Thus, the faster we have access to some information and the more detailed it is, the better we are able to respond in an appropriate way. Search and detection systems are built to provide such information. We can find such systems almost everywhere, for example, from a single soldier with binoculars, to remote sensors on the ocean floor up to sophisticated robot systems and unmanned aerial vehicles (UAVs). Information-gathering systems are imperfect and it takes time to find a target. There are two obvious ways to increase the system's performance: (1) make the technology better, and/or (2) develop better search algorithms and decision rules. There is a third way that is not so obvious and barely studied—changing the search environment.

This thesis focuses on analyzing the effect of the system's performance when we can influence the environment and if we are able to change it to cause an advantage for the searcher or a disadvantage for the adversary.

A simple, pictorial example is a building containing a hostile combatant. The task is to find him as soon as possible. We have to decide the order in which to search the floors and rooms and are constrained by the building's structure. Now, assume we can break a wall to reach another room more quickly. The question is to determine, which wall to break down to minimize the expected time to find the target.

1.2 Objectives

We will model the search environment as a simple connected undirected graph, which is a common method for a variety of search problems. By inserting new edges to the graph we are going to change the search environment's model. Our objective is to maximize search performance, that is to minimize the time needed to find the target. Considering a stationary target and a given initial graph, we want to determine, which edge to add to minimize the expected time to find a target inside the search environment. If we know something about the target's behavior and we have some idea where the target is more likely to be, we are going to include this information in the decision process. We want to answer the question: how does a prior probability distribution of the target's location affect the decision about the edges to add? If we allow the target to move, it raises the question of how this influences the decision. In particular, how do we handle target movement while updating the probability map, that is each cell's likelihood of containing the target? Extensive research on Markovian targets exists in the literature, as described in the next section. Does the solution for minimizing the expected search time by choice of edges to add still hold for a moving target? In addition to finding the target as fast as possible, we may want to constrain the target's movement. Which existing edge in the graph should we remove to make it as hard as possible for the target to escape detection?

To put our objectives in an operational scenario, assume we have to plan the search route for an unmanned aerial vehicle (UAV). The task is to find a ground target. We use a discretized search area and two search points are connected if they are accessible from each other in the original environment, i.e., a road in the search environment will result in a path of search nodes in the discretized model. We assume an imperfect sensor; in particular it could miss the target and thus has to search the area more than once. The first question is: which connections between search points should we artificially include in the model to minimize the time to find the target. Now assume we have a probability overlay for our discretized search model that provides information about the likelihood of target locations. We want to know if this additional information influences the choice of edges to add. Can we find a better solution using this additional information? In particular, can we find the target in shorter time? Intuitively, we will do best visiting the search points in order of probability from highest to lowest. But it takes time to travel from one point to another and we could spend too much time traveling. So, we are likely not interested in connections between points far away from each other. In the next step we allow the target to move. Do we have to adapt our model to that new situation in order to minimize the search time? Assume we could locate a ground team in the search area, i.e., a checkpoint.

We constrain the target's mobility by taking out an edge of the model without influencing the UAV's agility. Where should we place the checkpoint to most constrain the target's movement? If we could disconnect the search model we would create two smaller subproblems.

1.3 Literature Review

Graphs provide simple and pictorial ways to represent relationships between objects. For that reason, graph theory is widely adaptable to diverse problem solving. Networks, e.g., the Internet, social networks, biological networks, and simple road maps, are probably the most well-known examples that use graphs for modeling purposes. A small part of graph theory is focused on random graphs [1]. Although not obvious at first, random graphs are used to get insight into real-world problems [2, 3]. The first theoretical models of a network incorporating random graphs was proposed by Paul Erdös and Alfréd Rényi [4, 5]. Among all graphs, random or not, connected graphs are of special interest. Ensuring connectivity while generating random graphs is not an easy task and [6] provides a probabilistic investigation.

The Laplacian matrix of a graph and its eigenvalues can be used in various areas of mathematics with interpretations in several problems in physics, electrical engineering and many others. Among all eigenvalues of the graph Laplacian, one of the most popular is the second smallest. This particular eigenvalue has been studied extensively by M. Fiedler [7, 8]. In terms of graph theory, this eigenvalue is denoted as algebraic connectivity. Because of its importance and in honor of Fiedler's work, the associated eigenvector is known as Fiedler vector. Classifications of bounds to other graph properties as a function of algebraic connectivity are shown in [9] and recently summarized in [10].

While the strategy of assigning edge weights to graphs in order to maximize the algebraic connectivity has been widely studied and applied to various problems [11, 12, 13], the problem of adding some non-existing edges to achieve the same goal remains challenging. In fact, this problem is proved to be NP-hard [14]. In this research area, Boyd's publication [15] is of particular interest. He and his co-workers have identified and given a common framework for a lot of these edge-weight problems using a semidefinite program (SDP) [16]. For each of the missing edges, we can either add or not add that edge, which is a boolean decision variable. Thus, the SPD is constrained to a boolean solution. In [17], Boyd and Ghosh introduce a relaxation to the boolean constrained semidefinite program, as well as an application of the Fiedler vector. While the relaxed SDP does provide an optimal non-integer solution, which represents an upper bound for the augmented graphs algebraic connectivity, it does not directly yield a solution to the original problem. At the end, a simple rounding of the edge weights makes this a heuristic method. Neither the Fiedler vector method nor the relaxed SDP guarantees an optimal solution. In case of growing a graph by one edge, Kim [18] defines a method that uses the Fiedler vector and provides the optimal solution.

The connection between algebraic connectivity and search performance has been studied in [19], and a framework for decision making in probabilistic search can be found in [20]. To the best of our knowledge there is no publication that provides insight into the relation between augmentation of a given connected graph and search performance.

Another property we looked at during this study is the hitting time related to random walks on connected graphs [21, 22]. Calculating the hitting times for a given graph is computationally expensive, especially if the graph gets large. Markov chains have been used for that calculation and sophisticated algorithms have been developed to reduce the computational effort [23, 24]. The hitting time bridges directly from graph theory to search theory.

Search and search theory constitute a broad subject. The literature provides countless sources and applications with search problems. Many of those references, even if not directly applicable to our problem, help to learn about different techniques of problem solving. A variety of sequential optimization problems can be expressed as special cases of an elementary search problem [25]. Search on graphs is often related to dynamic programming [26], with the desire to find a self-adjusting data structure, reorder nodes to reveal similarities [27] or combinatorial search to reconstruct hidden graphs [28]. The author of [29] defines a small-world topology in a graph-theoretic sense and finds that such topologies can make a search problem very difficult and that the cost of solving such search problems can have a heavy-tailed distribution. In economics, we find another major field of search problems and applications. Defining the best out of several possible alternatives with different properties is considered a search problem [30] with very strong connections to optimization problems.

Search and detection from a military point of view typically looks for a lost object or a target. The problem variety ranges from searching for objects hidden in boxes [31] to searching for moving targets [32, 33], to pursuit-evasion problems [34, 35, 36, 37, 38]. Many search algorithm and techniques make use of Bayes' law to calculate and update probabilities of finding the target at a specific location [39, 40]. Those probabilities are used to find an optimal path through the search environment. Branch-and-bound algorithms used to solve search problems, as well as two classical search problems, can be found in [41].

1.4 Scope

Among the various kinds of search problems, we concentrate this study on a single searcher and a stationary target within a discretized environment.

Because of the known relationship between algebraic connectivity and search performance, we first study how to optimize the connectivity of a given graph by adding a specific number of edges. To measure the connectivity, we use the eigenvalues of the graph Laplacian, in particular the second-smallest eigenvalue.

Second, we analyze the effect of the graph modification on the search performance, assuming the searcher follows a computed path determined by a recursively updated likelihood of the target presence in each of the search cells—the probability map.

1.5 Main Contributions

With this study, we seek to enhance the relationship between graph and search theory. We can show that it is beneficial to add non-existing edges to the graph in order to increase search performance. The change in time to find the target caused by augmenting the graph follows an exponential pattern. This further implies that the gain in search performance decreases with graph density. We propose a formula that can be used to roughly approximate the decrease in time to find the target by adding edges to a sparse graph.

In addition, we analyze different methods to generate random simple, connected graphs as well as various methods to grow a graph. We rigorously evaluate these various methods and compare them with existing approaches. Further, we provide a model that incorporates probabilistic target information with the search problem.

1.6 Organization of Thesis

In Chapter 2, we summarize the known results related to the problem we try to solve. We subdivide the review into subsection probabilistic search, graph theory, and optimization. Furthermore, we define the notions used later on in this chapter.

Chapter 3 starts with the analysis of the different methods to generate a random simple, connected graph, followed by finding the best way to grow a graph for the purpose of decreasing search time. That decision is supported by simulation results. Finally, we propose a formula to roughly approximate the gain of adding edges. Analysis is followed immediately by results in

Chapter 3 for all the small subproblems in the same chapter.

Furthermore, we briefly describe problems we have seen during this study related to search behavior, false detection rates, and projection of the results to more realistic problems that could appear in real life. Those problems are considered in ideas for future work in Chapter 4.

CHAPTER 2: MATHEMATICAL FORMULATION

2.1 Problem Formulation

This thesis' goal is to alter the search environment in order to maximize search performance. We use the ***time to decision (TTD)*** as measure of performance (MOP). The search is conducted on graph $\mathcal{G}$ with vertex set $\mathcal{V}$, edge set $\mathcal{E}$ and with initial probability distribution $p(v)$, that is the likelihood of the target's presence at vertex v. We use the symbol $TTD(\mathcal{G}(\mathcal{V}, \mathcal{E}), p(v))$ or simplified $TTD(\mathcal{G}, p(v))$ to denote the associated search performance. The searcher is considered as given and we are not able to change any of its properties such as sensor characteristics. The environment, modeled as a simple connected graph, can be modified by adding k, $0 \leq k \leq \overline{m}$, non-existing edges to the graph $\mathcal{G}$. The set of non-existing edges is denoted by $\overline{\mathcal{E}}$, with $|\overline{\mathcal{E}}| = \overline{m}$. The solution, i.e., the set of edges to add in order to best increase the search performance, is clearly a subset of $\overline{\mathcal{E}}$ and is denoted by the symbol $\mathcal{E}^*$. The problem can be stated as

$$\begin{aligned} \min \quad & TTD\left(\mathcal{G}(\mathcal{V}, \mathcal{E} \cup \mathcal{E}^*), p(v)\right) \\ s.t. \quad & |\mathcal{E}^*| = k, \\ & \mathcal{E}^* \subseteq \overline{\mathcal{E}}. \end{aligned}$$

Because of the relationship between algebraic connectivity, λ_2 (the second-smallest eigenvalue of the graph Laplacian, $\mathcal{L}$) and search performance, as shown in [19], we want to solve the problem

$$\begin{aligned} \max \quad & \lambda_2\left(\mathcal{L}(\mathcal{G}(\mathcal{V}, \mathcal{E} \cup \mathcal{E}^*))\right) \\ s.t. \quad & |\mathcal{E}^*| = k, \\ & \mathcal{E}^* \subseteq \overline{\mathcal{E}} \end{aligned}$$

in the first step and analyze the effect to the search performance in a second step.

2.2 Probabilistic Search Review

The primary question in search is to determine where the target is located in a region $\mathcal{A}$. We want to know the target's exact position or at least narrow it down to a small part of the search region. If the target is considered stationary and is in fact located somewhere in the region, we have a good chance to answer the question. Depending on the searcher's properties, it takes more or less effort, but at the end we will find the target assuming no constraint on search effort or resources (e.g., time). The effort needed to come up with an answer can be unbounded if our initial ***belief*** in the target's presence is incorrect and the target is not located inside the search region. We need to provide a decision rule for the searcher as to when to stop the search without determining the target's position. The primary question becomes whether or not the target is present in the search region $\mathcal{A}$, and if so, the secondary question is about its position.

2.2.1 Definitions and Notations

The uncertainty of the target's presence or absence in the search region can be considered a binary hypothesis test and expressed as H, a Bernoulli random variable; i.e., H can take either zero or one. The probability that the target is present becomes $P(H = 1)$. Consider a discretized search region $\mathcal{A}$, with cells $c \in \{1, \ldots, |\mathcal{A}|\}$. In addition, we define a virtual cell $\emptyset$ that represents all space outside the search environment. A target can be either in one of the search region cells or in the virtual cell, which represents its absence from the search region. We define $p_\emptyset \overset{\text{def}}{=} P(H = 0) = 1 - P(H = 1)$. The presence in one of these cells, also known as the ***cell belief***, is another Bernoulli random variable, denoted as X_c or $X_\emptyset$. At the beginning of the search process (time, $t = 0$) there may be a particular probability that the target is inside the c^{th} search cell, $p_c^0 \overset{\text{def}}{=} P\left(X_c = 1\right)$ or outside the search region, $p_\emptyset^0 \overset{\text{def}}{=} P\left(X_\emptyset = 1\right) = 1 - \sum_{c=1}^{|\mathcal{A}|} p_c^0$. In the case of the absence of such prior cell beliefs, the cells are weighted by a noninformative distribution (i.e., uniform) and $p_c^0 = \frac{1}{|\mathcal{A}|},\ \forall c \in \mathcal{A}$. Figure 2.1 and Figure 2.2 depict the idea of the initial target location using initial probability maps and initial belief.

The single searcher's position, $s(t)$, will be in one of the search region's cells at any time: $s(t) \in \{1, \ldots, |\mathcal{A}|\},\ \forall t \in \{1, 2, \ldots\}$. We can define another Bernoulli random variable, $Y_{s(t)}$, that determines whether the searcher detects the target in the cell at time t. We want to take into account that the searcher's sensor is imperfect. We allow the searcher to make ***false positive*** detections, that is, detection of a target that is in fact not present, and ***false negative*** detections, which is a missed detection of a present target, with probabilities

$$P\left(Y_c = 1 \middle| X_c = 0\right) \overset{\text{def}}{=} \alpha \qquad \text{(false positive)},$$

$$P\left(Y_c = 0 \middle| X_c = 1\right) \overset{\text{def}}{=} \beta \qquad \text{(false negative)}.$$

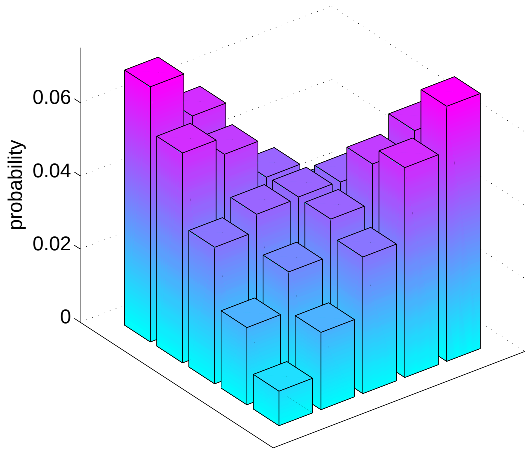


Figure 2.1: **Initial probability map with belief one** - Arbitrary initial cell belief for a belief of one, e.g. we have perfect information that the target is located somewhere inside the search region. All probabilities sum up to 1.

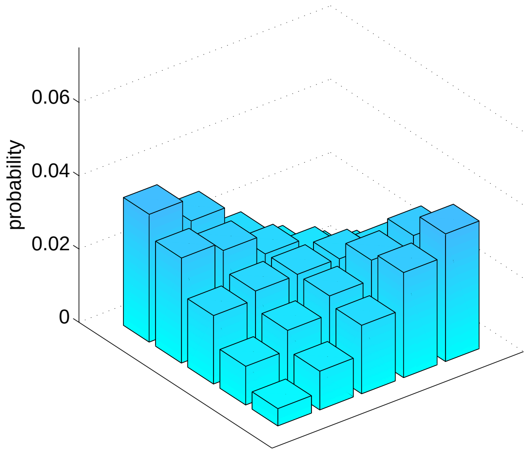


Figure 2.2: **Initial probability map with belief one half** - The same initial cell beliefs for a belief of one half, e.g. we have no information whether the target is located inside the search region. The probabilities sum up to $\frac{1}{2}$. The remaining probability mass is allocated to the virtual cell.

Applying the law of total probability and Bayes' theorem [42]:

$$P(B) = P(B|A_1)P(A_1) + \ldots + P(B|A_k)P(A_k) = \sum_{i=1}^{k} P(B|A_i)P(A_i),$$

$$P(A_j|B) = \frac{P(A_j \cup B)}{B} = \frac{P(B|A_j)P(A_j)}{\sum\limits_{i=1}^{k} P(B|A_i)P(A_i)} \qquad j = 1, \ldots, k$$

we can update the probabilities of the target's presence for all cells c after the t^{th} time steps:

$$p_c^t = \begin{cases} P\left(X_c^t = 1 | Y_{s(t)} = 1\right) = \dfrac{(1-\beta)\, p_c^{t-1}}{(1-\beta)\, p_c^{t-1} + \alpha\,(1 - p_c^{t-1})} & \text{if } s(t) = c,\ Y_{s(t)} = 1 \\ P\left(X_c^t = 1 | Y_{s(t)} = 0\right) = \dfrac{\beta p_c^{t-1}}{\beta p_c^{t-1} + (1-\alpha)\,(1 - p_c^{t-1})} & \text{if } s(t) = c,\ Y_{s(t)} = 0 \\ P\left(X_c^t = 1 | Y_{s(t)} = 1\right) = \dfrac{\alpha p_c^{t-1}}{(1-\beta)\, p_{s(t)}^{t-1} + \alpha \left(1 - p_{s(t)}^{t-1}\right)} & \text{if } s(t) \neq c,\ Y_{s(t)} = 1 \\ P\left(X_c^t = 1 | Y_{s(t)} = 0\right) = \dfrac{(1-\alpha)\, p_c^{t-1}}{\beta p_{s(t)}^{t-1} + (1-\alpha)\left(1 - p_{s(t)}^{t-1}\right)} & \text{if } s(t) \neq c,\ Y_{s(t)} = 0 \end{cases}$$

The author in [19] provides an expression that combines these four cases in one single equation (2.1). He defines

$$\begin{aligned}\Phi\left(Y_{s(t)}\right) &\overset{\text{def}}{=} \left(1 - Y_{s(t)}\right)(1-\alpha) + Y_{s(t)}\alpha, \\ \Psi\left(Y_{s(t)}\right) &\overset{\text{def}}{=} \left(1 - Y_{s(t)}\right)\beta + Y_{s(t)}(1-\beta),\end{aligned}$$

and finally

$$p_c^t = \frac{\Theta_c\left(Y_{s(t)}\right) \cdot p_c^{t-1}}{\Phi\left(Y_{s(t)}\right) + \Omega\left(Y_{s(t)}\right) \cdot p_{s(t)}^{t-1}}. \tag{2.1}$$

The result is a compact and easy way to update the probability map

$$\begin{aligned}\text{with}\quad \Omega\left(Y_{s(t)}\right) &\overset{\text{def}}{=} \Psi\left(Y_{s(t)}\right) - \Phi\left(Y_{s(t)}\right) = \left(2Y_{s(t)} - 1\right)(1-\alpha-\beta) \\ \text{and}\quad \Theta_c\left(Y_{s(t)}\right) &\overset{\text{def}}{=} \begin{cases} \Psi\left(Y_{s(t)}\right), & \text{if } s(t) = c \\ \Phi\left(Y_{s(t)}\right), & \text{if } s(t) \neq c. \end{cases}\end{aligned}$$

Recall that $s(t)$ is the searcher's location at time t and c is the cell whose probability is being updated. A recursive Bayesian update gives the Equation 2.2.

$$p_c^t = \frac{\prod_{k=1}^{t} \Theta_c\left(Y_{s(k)}\right) p_c^0}{\prod_{k=1}^{t} \Phi\left(Y_{s(k)}\right) + \sum_{k=1}^{t} \left(\prod_{l=1}^{k-1} \Theta_{s(k)}\left(Y_{s(l)}\right)\right) \Omega\left(Y_{s(k)}\right) \left(\prod_{m=k+1}^{t} \Phi\left(Y_{s(m)}\right)\right) p_{s(k)}^0} \tag{2.2}$$

Similarly, the probability that the target is outside the search region can be expressed and updated by

$$p_\emptyset^t = \frac{\Phi\left(Y_{s(t)}\right) \cdot p_\emptyset^{t-1}}{\Phi\left(Y_{s(t)}\right) + \Omega\left(Y_{s(t)}\right) \cdot p_{s(t)}^{t-1}} \tag{2.3}$$

$$= \frac{\prod_{k=1}^{t} \Phi\left(Y_{s(t)}\right) p_\emptyset^0}{\prod_{k=1}^{t} \Phi\left(Y_{s(k)}\right) + \sum_{k=1}^{t} \left(\prod_{l=1}^{k-1} \Theta_{s(k)}\left(Y_{s(l)}\right)\right) \Omega\left(Y_{s(k)}\right) \left(\prod_{m=k+1}^{t} \Phi\left(Y_{s(m)}\right)\right) p_{s(k)}^0}. \tag{2.4}$$

Equations 2.1–2.4 are needed to determine the best search path in order to minimize the expected time until detection. Furthermore, we need to update the probability that the target is present, which is an easy task if we have found the target. In case there is no detectable target we need to decide whether to stop or proceed with the search. This is related to the problem of (optimal) stopping criterion found in the search literature [43, 44]. The probability mass accumulated in the virtual cell is the indicator for the target's presence. Dealing with imperfect sensors, this value is strictly greater than 0 and strictly less than 1 ($0 < p^t_\emptyset < 1|\alpha, \beta \neq 0$). For that purpose, we introduce a lower (B_l) and upper (B_u) probability threshold. The current belief in the target's presence is defined as $B(t) = 1 - p^t_\emptyset$ and gets updated after every cell search. If the cell belief $p^t_c > B_u$ for any cell c, we believe the target is present and it is located at cell c. Note that this cell is not necessarily the current searcher's position. In this case, the belief $B(t)$ exceeds the upper threshold B_u. If $B(t) < B_l$, thus $\sum_c p^t_c < B_l$, the searcher will stop the search process and the search area is considered target free. The decision made by the preceding rules may be wrong in both cases. Figure 2.3 depicts the idea of the introduced thresholds and shows the different belief traces for a perfect and an imperfect searcher.

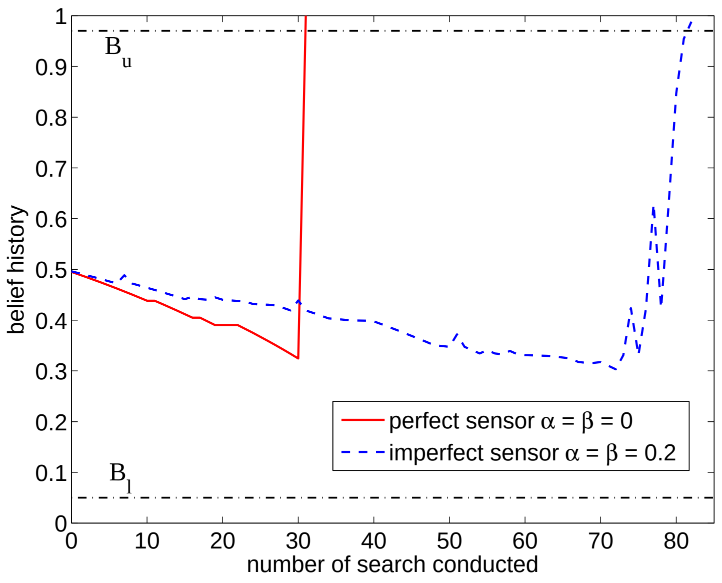


Figure 2.3: **Belief of target presence** - Searches with different searcher were conducted on a graph with n=50 and m=60. The initial probability map was uniform with belief $\frac{1}{2}$. Target was stationary at the same location for both searches as well as the initial searcher's position. The thresholds were set to be B_l=0.05 and B_u=0.97.

2.2.2 Some Search Behaviors

Finally, we put the searcher in one of the search region's cells, the start cell. Besides the initial belief of target presence, $B(0)$, and the initial probability mass, p^0_c, $\forall c \in \{1 \ldots |\mathcal{A}|\}$ (i.e., the initial ***probability map***) the searcher needs to be equipped with some rules to conduct the search, that is, the ***search behavior***. We know from search theory that a random search is the lower bound to compare against any other search algorithm [45, 46].

For this thesis, we investigate two different search behaviors, both of which restrict the searcher's motion to only neighboring cells. The ***myopic*** searcher will move to the neighbor (possibly the current position) with highest probability that the target is present. Thus, if the searcher's current cell is weighted with the highest probability among all of its possible neighbors, the searcher will conduct the next search at the same position. Motivated by the fact that the incremental change in belief is monotonically increasing with increasing search-cell probability mass, a ***shortest path*** behavior can be defined. The maximum-probability cell is defined by the probability map. Planning the shortest path for transiting from the searcher's current position to that maximal probability cell can be done by efficient algorithms, such as Dijkstra's. On its path, the searcher is still constrained by the environment and thus will usually visit several other cells before reaching the highest probability cell. At each of the visited cells, it will conduct a search but proceed on the planned path regardless of the search result. This form of search behavior is called "semi-adaptive" search [47]. If the searcher cannot define the next step uniquely by the given rules and has to choose from more than one equal option, it will randomly choose one of them. This is true for both search behaviors.

2.3 Graph-Theory Review

Graphs are mathematical structures to model relations between objects and have been extensively studied in the past. Graph theory has been widely adapted to diverse problems since its first appearance, and for that reason we can find various different notations. We adopt the notations from [48], and provide an overview in this section.

2.3.1 Definitions and Notations

We are going to model the search environment as a graph. $\mathcal{G} = (\mathcal{V}(\mathcal{G}), \mathcal{E}(\mathcal{G}))$ where $\mathcal{V}(\mathcal{G}) = \{v_1, \dots, v_n\}$ is called the ***vertex set*** with $n = |\mathcal{V}|$, and $\mathcal{E}(\mathcal{G}) = \{e_{ij}\}$ is called the ***edge set*** with $m = |\mathcal{E}|$. An edge, e_{ij}, connects vertices v_i and v_j if they are adjacent or neighbors. We say v_i and v_j are ***adjacent*** if $e_{ij} \in \mathcal{E}(\mathcal{G})$. The number of neighbors of a node v is called the ***degree*** of v and is denoted by $deg(v)$. We denote the largest degree with $\Delta(\mathcal{G})$ and the smallest degree with $\delta(\mathcal{G})$. We consider a ***simple graph***, without multiple edges or loops.

The ***complement*** of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is $\overline{\mathcal{G}} = (\mathcal{V}, \overline{\mathcal{E}})$. The set $\overline{\mathcal{E}}(\mathcal{G}) = \{e_{ij} | e_{ij} \notin \mathcal{E}(\mathcal{G})\}$ is the set of all possible edges not in the edge set of $\mathcal{G}$, with $\overline{m} = \left|\overline{\mathcal{E}}\right|$ and $\mathcal{G}(\mathcal{V}, \mathcal{E} \cup \overline{\mathcal{E}}) = K_n$, the complete graph.

A u-w ***walk*** in $\mathcal{G}$ is a sequence of vertices in $\mathcal{G}$, beginning with u and ending at w, such that consecutive vertices in the sequence are adjacent; that is, we can express the walk as $u =$

$v_1, v_2, v_2, \ldots, v_k = w$. A u-w walk in a graph in which no vertices are repeated is a ***path***. If $\mathcal{G}$ contains a u-v path, then u and v are said to be connected. A graph $\mathcal{G}$ is ***connected*** if every pair of vertices of $\mathcal{G}$ are connected, that is, if $\mathcal{G}$ contains a u-v path for every pair u, v of distinct vertices of $\mathcal{G}$.

The ***distance*** between u and v is the smallest length of any u-v path in $\mathcal{G}$. The greatest distance between any two vertices of a connected graph $\mathcal{G}$ is called the ***diameter*** and is denoted by $diam(\mathcal{G})$.
For a graph with n vertices, the entries of the $n{\times}n$ adjacency matrix Adj are defined by:

$$Adj_{ij} := \begin{cases} 1, \text{ if there is an edge } e_{ij} \\ 0, \text{ if there is no edge } e_{ij} \\ 0, \text{ if } i = j \end{cases}$$

A simple connected graph with n vertices has at least $(n-1)$ edges and at most $\frac{1}{2}n(n-1)$ edges, in the latter case the graph is complete. A graph S is called a ***subgraph*** of $\mathcal{G}$, written $S \subseteq \mathcal{G}$, if $\mathcal{V}(S) \subseteq \mathcal{V}(\mathcal{G})$ and $\mathcal{E}(S) \subseteq \mathcal{E}(\mathcal{G})$. A subgraph F of the graph $\mathcal{G}$ is called an ***induced subgraph*** of $\mathcal{G}$ if whenever u and v are vertices of F and $e_{uv} \in \mathcal{E}(\mathcal{G})$ then $e_{uv} \in \mathcal{E}(\mathcal{F})$ as well. To emphasize this is an induced subgraph of $\mathcal{G}$, we denote this subgraph by $\langle S \rangle_{\mathcal{G}}$.

2.3.2 The Graph Laplacian

The graph ***Laplacian***, $\mathcal{L}(\mathcal{G})$ is a $n{\times}n$ matrix of the form $\mathcal{L} := Deg - Adj$ where Deg is the degree matrix, i.e.,

$$Deg_{ij} := \begin{cases} deg(v_i), & \text{if } i = j \\ 0, & \text{o.w.} \end{cases}$$

Another way to express the graph Laplacian is using the ***edge vector*** $a_e \in \mathbb{R}^n$, which is defined for any $e = e_{ij}$ by

$$a_{e_k} := \begin{cases} 1, \text{ if } k = i \\ -1, \text{ if } k = j \\ 0, \text{ o.w.} \end{cases}$$

The matrix $a_e a_e^T$ is the graph Laplacian for the graph with the edge e, $\mathcal{G} = (\mathcal{V}, \{e\})$. Thus, the graph Laplacian for $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be expressed as

$$\mathcal{L} = \sum_{e=1}^{m} a_e a_e^T.$$

The eigenvalues of the graph Laplacian are of interest for many applications—here in particular the second-smallest eigenvalue, which is the algebraic connectivity [7]. The eigenvalues, λ, are defined by:

$$\begin{aligned} \mathcal{L}x &= \lambda x \\ (\mathcal{L} - \lambda \mathbf{I})x &= 0 \end{aligned}$$

which has a nontrivial solution if $det(\mathcal{L} - \lambda \mathbf{I}) = 0$, where $\mathbf{I}$ is the identity matrix. The ***algebraic connectivity***, λ_2, has some interesting properties [7, 9]:

- λ_2 is monotonically non-decreasing with adding edges to the graph $\mathcal{G}$,
- $\lambda_2 = 0$ if and only if $\mathcal{G}$ is not connected,
- $\lambda_2 = |\mathcal{V}|$ when $\mathcal{G}$ is a complete graph and
- $\lambda_2 \leq \delta(\mathcal{G})$.

The corresponding eigenvector to the second-smallest eigenvalue is called the Fiedler vector. Note that the smallest eigenvalue for a connected graph is always zero.

2.3.3 On Random Walks

The notion of random walks comes into play for the search problem if we have no idea where the target could be located. The initial probability distribution across the nodes is uniform, thereby the search starts as a random walk.

Given a graph and a starting point, a searcher selects a neighboring cell at random, moves to this neighbor and selects a neighbor from this position, etc. The sequence of vertices produced this way is a ***random walk*** on the graph. A random walk can be considered a finite Markov chain. There is not much difference between the theory of random walks on graphs and finite Markov chains. Various aspects of the theory of random walks on graphs are surveyed [22]. In particular, hitting times and return times are of interest for our study.
The access time or ***hitting time***, H_{ij}, is the expected number of steps it takes in a random walk

to visit node j, starting at node i. The sum of the hitting times $H_{ij} + H_{ji}$ is called the ***commute time***, the expected number of steps in a random walk to visit node j and return to node i, starting at node i. The ***return time*** for a node j is the expected time until a searcher revisits that node without any constraints. So we do not care if any other node is visited before we hit the start node again. We find the solution by solving the Markov chain for the long-run proportion of being in a specific state.

2.4 Optimization Review

We want to find the best edge (or a number of best edges) to add to an existing graph in order to maximize search performance. Our measure of performance is the time it takes to make the decision whether the target is present or absent. We have to solve an optimization problem. According to the problem setup, optimization methods are often divided into several subfields, i.e., linear programming [49], non-linear programming [50], and convex optimization including semidefinite programming [16]. Among the different and problem-specific algorithms, we are interested in exact algorithms that provide optimal solutions or algorithms in which we can define the error magnitude. Unfortunately, this is not always achievable, or we have to pay with a huge amount of computational effort. For this reason, knowing we will be able to do better, we sometimes use heuristic algorithms that provide good solutions.

2.4.1 Brute Force

This method simply tries all possible combinations of edges to add and thus is an exact algorithm. It is guaranteed to find the optimal solution. Unfortunately, it takes $\binom{|\overline{\mathcal{E}}|}{k}$ eigenvalue computations, where $|\overline{\mathcal{E}}|$ is the number of possible edges to add and k is the actual number of edges to add. We used this method for comparison purposes for adding up to two edges. With an increasing number of nodes, the number of calculations to conduct increases dramatically. The curse of dimensionality renders brute force improper for practical use.

2.4.2 Fiedler Vector Method

This method uses the Fiedler vector, i.e., the eigenvector associated with the second-smallest eigenvalue, to find an edge to add. It was first introduced as a greedy perturbation heuristic in [17]. We add the k edges sequentially, one at a time, choosing the edge e_{ij} from the remaining candidate set with the largest value $(v_i - v_j)^2$, where v is the Fiedler vector of the current Laplacian. We need to conduct k eigenvector computations to find k edges to add. One eigenvalue and eigenvector calculation takes approximately $O(\frac{4}{3}n^3)$ operations. If v is the unit eigenvector corresponding to λ_2, then vv^T is a supergradient of λ_2 at L. When $\lambda_1{<}\lambda_2{<}\lambda_3$, then vv^T gives the first order approximation of the increase in λ_2. So, the motivation for this method is to add the edge e_{ij} that gives the largest predicted increase in $\lambda_2(L)$, according to a first-order approximation.

This method does pretty well in many cases, but does not guarantee an optimal solution, either for adding one edge or for the sequence of edges. In [18], this issue is reviewed and an algorithm is given that guarantees the optimal solution for adding one edge. The proposed algorithm incorporates λ_3, the third-smallest eigenvalue of the graph Laplacian, in addition to the algebraic connectivity. A bisection technique is used to stepwise narrow down the list of possible edges to add. The method increases the computational complexity to $O(5.7mn)$, compared to $O(2mn)$ for the greedy-type heuristic, excluding the eigenvector calculation. The given algorithm requires graphs that have the two eigenvalues different, $\lambda_2 \neq \lambda_3$, involves much more computational effort, and conducting this method sequentially does not provide optimality. We would end up with another heuristic approach. In this thesis, we are not following this path.

2.4.3 Semidefinite Program (SDP)

In his 1973 paper [7], Fiedler has already stated and proven that $\tilde{M} = M - \lambda_2\left(\mathbf{I} - n^{-1}\mathbf{J}\right)$ is semidefinite for any $n{\times}n$ semidefinite matrix M with corresponding second-smallest eigenvalue λ_2, where $\mathbf{I}$ is the identity matrix and $\mathbf{J}$ the matrix of all ones. The graph Laplacian $\mathcal{L}$ with $\lambda_i \geq 0,\ i = 1, 2, \ldots, n$ is always positive semidefinite.

The problem of how to maximize the algebraic connectivity of the graph by adding k, $0 \leq k \leq \overline{m}$, edges to the graph can be formulated as

$$\begin{aligned} \max \quad & \lambda_2\left(\mathcal{L}(\mathcal{G}(\mathcal{V}, \mathcal{E} \cup \mathcal{E}^*))\right) \\ s.t. \quad & |\mathcal{E}^*| = k, \\ & \mathcal{E}^* \subseteq \overline{\mathcal{E}}. \end{aligned}$$

The decision variable for the maximization problem, whether to include an edge or not, is denoted by $x \in \{0,1\}^{\overline{m}}$, where $x_e = 1$ if an edge belongs to the subset $\mathcal{E}^*$[17]:

$$\begin{aligned} \max \quad & \lambda_2 \left(\mathcal{L} + \sum_{e=1}^{\overline{m}} x_e a_e a_e^T \right) \\ s.t. \quad & \mathbf{1}^T x = k, \\ & x \in \{0,1\}^{\overline{m}}. \end{aligned}$$

We introduce an shortcut for the resulting graph Laplacian $\mathcal{L}(x)$ and relax the problem by removing the boolean constraint for the decision variable x, obtaining

$$\mathcal{L}(x) = \mathcal{L} + \sum_{e=1}^{\overline{m}} x_e a_e a_e^T$$

with relaxation

$$\begin{aligned} \max \quad & \lambda_2 (\mathcal{L}(x)) \\ s.t. \quad & \mathbf{1}^T x = k, \\ & 0 \le x \le 1. \end{aligned}$$

Boyd used the semidefinite property to formulate a semidefinite program in [17] to maximize the algebraic connectivity by adding edges to the graph:

$$\begin{aligned} \max \quad & s \\ s.t. \quad & s \left(\mathbf{I} - \mathbf{1}\mathbf{1}^T / n \right) \preceq \mathcal{L}(x) \\ & \mathbf{1}^T x = k, \\ & 0 \le x \le 1. \end{aligned} \tag{2.5}$$

To see why this is true, let us first decompose $\mathcal{L}(x) = U \Lambda U^{-1}$, where Λ is the matrix with all eigenvalues of $\mathcal{L}(x)$ at the diagonal and 0 otherwise. U is the matrix with all corresponding eigenvectors. Because $\mathcal{L}(x)$ is symmetric, U is orthogonal, and therefore $U^{-1} = U^T$. Thus, $\mathcal{L}(x) = U \Lambda U^T$. We know that the smallest eigenvalue of the graph Laplacian is always 0 and the corresponding eigenvector is $\mathbf{1}$. A normalized U looks like this:

$$U = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n-1} & 1/\sqrt{n} \\ a_{21} & a_{22} & \cdots & a_{2n-1} & 1/\sqrt{n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-11} & a_{n-12} & \cdots & a_{n-1n-1} & 1/\sqrt{n} \\ a_{n1} & a_{n2} & \cdots & a_{nn-1} & 1/\sqrt{n} \end{pmatrix},$$

where the product of rows x and y is

$$a_{x1}a_{y1} + \ldots + a_{xn-1}a_{yn-1} + \frac{1}{\sqrt{n}}\frac{1}{\sqrt{n}} = \begin{cases} 1, & \text{if } x = y \\ 0, & \text{if } x \neq y. \end{cases} \tag{2.6}$$

Second, let's find the eigenvalues of $\left(\mathbf{I} - \mathbf{1}\mathbf{1}^T/n\right)$.

$$\left(\mathbf{I} - \mathbf{1}\mathbf{1}^T/n\right)\mathbf{v} = \lambda\mathbf{v} = \mathbf{I}\mathbf{v} - \frac{\mathbf{1}}{n}\mathbf{1}^T\mathbf{v}$$

It is certainly true $\exists\mathbf{v} \ni \mathbf{1}^T\mathbf{v} = \mathbf{0}$. This is an eigenvector associated with $\lambda = 1$ and clearly there are $n-1$ of these. The remaining eigenvector is $\mathbf{1}$ with associated eigenvalue $\lambda = 0$.

$$\left(\mathbf{I} - \mathbf{1}\mathbf{1}^T/n\right)\mathbf{1} = \lambda\mathbf{1} = \mathbf{1} - \frac{\mathbf{1}}{n}\mathbf{1}^T\mathbf{1} = \mathbf{1} - \frac{\mathbf{1}}{n}n = 0$$

Third, multiply $U\Gamma U^{-1} = U\Gamma U^T$ where U is the orthogonal matrix from the $\mathcal{L}(x)$ decomposition and Γ is the matrix with all the eigenvalues of $\left(\mathbf{I} - \mathbf{1}\mathbf{1}^T/n\right)$ at the diagonal and 0 otherwise:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n-1} & 0 \\ a_{21} & a_{22} & \cdots & a_{2n-1} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-11} & a_{n-12} & \cdots & a_{n-1n-1} & 0 \\ a_{n1} & a_{n2} & \cdots & a_{nn-1} & 0 \end{pmatrix} \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{n-11} & a_{n1} \\ a_{12} & a_{22} & \cdots & a_{n-12} & a_{n2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{1n-1} & a_{1n-1} & \cdots & a_{n-1n-1} & a_{nn-1} \\ 1/\sqrt{n} & 1/\sqrt{n} & \cdots & 1/\sqrt{n} & 1/\sqrt{n} \end{pmatrix}$$

$$= \begin{pmatrix} 1-1/n & -1/n & \cdots & -1/n & -1/n \\ -1/n & 1-1/n & \cdots & -1/n & -1/n \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -1/n & -1/n & \cdots & 1-1/n & -1/n \\ -1/n & -1/n & \cdots & -1/n & 1-1/n \end{pmatrix} = \left(\mathbf{I} - \mathbf{1}\mathbf{1}^T/n\right).$$

Because of Equation (2.6) we know

$$a_{x1}a_{y1} + \ldots + a_{xn-1}a_{yn-1} = \begin{cases} 1-1/n, & \text{if x = y} \\ -1/n, & \text{if x} \neq \text{y.} \end{cases}$$

Finally putting all the pieces together yields

$$\begin{aligned} s\left(\mathbf{I} - \mathbf{1}\mathbf{1}^T/n\right) &\preceq \mathcal{L}(x), \\ \mathcal{L}(x) - s\left(\mathbf{I} - \mathbf{1}\mathbf{1}^T/n\right) &\succeq 0, \\ U\Lambda U^T - sU\Gamma U^T &\succeq 0, \\ U(\Lambda - s\Gamma)U^T &\succeq 0. \end{aligned}$$

We constrain the left-hand side of the SDP to be positive semidefinite and subtract s from every eigenvalue of $\mathcal{L}(x)$, so s can be at most equal to $\lambda_2(\mathcal{L}(x))$. If we maximize s at the same time, we certainly maximize $\lambda_2(\mathcal{L}(x))$. This proves the correctness of (2.5). Good references for the linear-algebra proportion of the previous derivation are [51, 52, 53].

Because of the SDP relaxation, the result is an upper bound for the optimal integer solution. The number of edges to add is smeared over all the possible edges to add, i.e., the optimal solution contains fractions of edges. To solve our original problem, we are not interested in fractions of edges to add. We either add an edge or not. Once again we need a heuristic to determine, which edge to choose. The best we can do is simply rounding to find the k best edges, which is not guaranteed to be optimal.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3: SIMULATION RESULTS AND ANALYSIS

3.1 Generating Random Simple, Connected Graphs

The first step in the analysis is to find a search environment that will be modeled as a graph. To better understand the role of graph-theoretic properties on search performance, we want to include all kinds of graphs in our analysis. We do not have any requirements on a graph, other than that is a connected graph. We are in need for a way to quickly generate random simple connected graphs even with large node numbers. In this section, we present two ways to do so. One is computationally much faster than the other, but produces graphs with higher maximum degrees, Δ, whereas the second approach generates more uniform maximum degrees.

3.1.1 The Spanning-Tree Method

The result we want the random graph generator to return is a connected graph. Every connected graph has a ***spanning tree***, a subgraph containing $(n-1)$ edges that is connected. Conversely, a graph having such a spanning tree must be connected. That is the key for the spanning-tree method.

Suppose we want to generate a connected graph with n nodes and m edges. Initially, all the numbered nodes are in a "basket" representing available nodes. In a second basket we store the connected sub-spanning tree, currently empty. The algorithm picks randomly a node from the set of available nodes and a node from the connected sub-spanning tree. It places an edge between those two nodes and puts the node chosen from the available node basket into the basket with the sub spanning tree. Repeating this n times will leave the available node basket empty, and we find a spanning-tree with n nodes and $(n-1)$ edges in the other basket. We now have to choose $(m-n+1)$ edges from the remaining $\frac{n(n-1)}{2} - (n-1)$ possible edges and place those in the graph.

We have to conduct only one try to come up with a desired graph and we are able to specify the number of edges explicitly. The disadvantage is that we produce graphs that tend to have high maximum degrees (Δ). The node first chosen to start to build the sub-spanning tree is likely to have the most incident edges. We especially observe this for sparse graphs with larger node numbers.

3.1.2 The Acceptance-Rejection Method

This method allows an algorithm to generate random graphs that violate any of our requirements. In an additional step, the graph gets checked and accepted or rejected. Rejection would cause the entire algorithm to start again.

Again, suppose we want to generate a connected graph with n nodes and m edges. The number of possible edges is $\frac{n(n-1)}{2}$. The algorithm draws a binomial random variable with probability of success (p) for each of those possible edges. It puts an edge into the edge set $\mathcal{E}(\mathcal{G})$ if the coin flip succeeded. If $p = \frac{2m}{n(n-1)}$, than the expected number of edges in the resulting graph is m.

We calculate the second-smallest eigenvalue for the preliminary result and accept the graph if $\lambda_2 \neq 0$. Because it is most unlikely to get the exact number of desired edges in the graph, we will work with an lower (m_l) and upper bound (m_u) and accept the graph if $m_l \leq m \leq m_u$. That still enables us to constrain the number of edges to a specific number.

This method is unbiased in selecting edges and any node could possibly end up with the highest degree. The downside is that we have to do many more calculations. Sparse graphs are likely to violate the connectivity property, and the smaller ($m_u - m_l$) is, the longer it takes to find a graph.

3.1.3 Comparing the Methods by Maximum Degree

Besides the different computational effort required by the two methods already mentioned, the concern remains that the spanning-tree method produces graphs with higher maximum degree. This effect appears to be more pronounced for sparse graphs than for dense.

To show this, we generate random graphs with 50 nodes and restrict the number of edges to be within one of three different ranges. The ranges are [55, 65], [500, 520] and [1000, 1020]. We generate 50 graphs for each range by each generating method. If it happens that both methods return a graph with the same number of edges, we increment the count for the method, which graph has the higher largest degree. The result is shown in Figure 3.1. It is not surprising that the maximum degrees vary among different graphs with the same number of edges. For the middle- and high-density graphs, the plot looks like the result we expect from similar methods. In contrast, the sparse graphs look different. Here the spanning-tree method produced graphs with a mean maximum degree of 7.24 and a standard deviation of 1.17. The mean maximum degree for the acceptance-rejection method is 6.54, with a standard deviation of 1.01. This verifies our concern that the spanning-tree method produces graphs with higher maximum degree.

Because of this result, we elect to use the slower (but unbiased in maximum degree) acceptance–rejection method to generate random graphs.

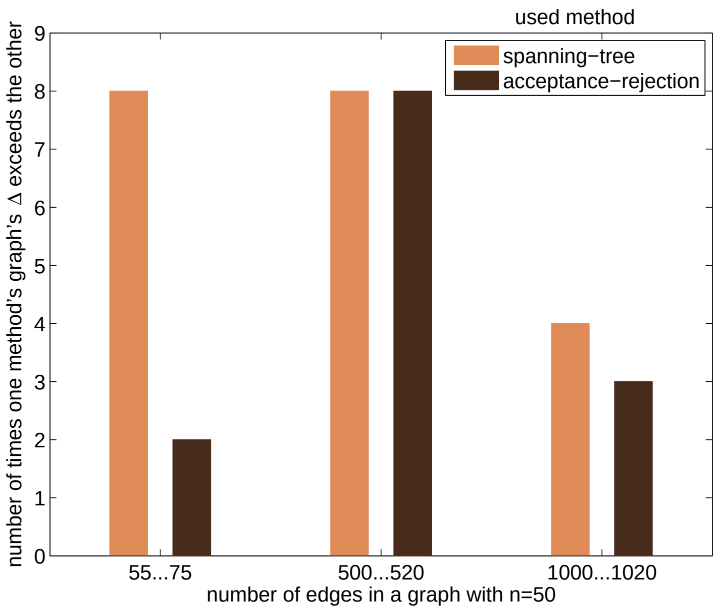


Figure 3.1: **Comparison of the two random graph-generating methods** - For each edge number range, 50 random graphs were generated by each method. The counts are the number of times that one method's graph's maximum degree is bigger than the other's maximum degree, provided the number of edges are the same. The maximum-degree range was $[5, 9]$ with $\mu = 6.54$, $\sigma = 1.01$ for the acceptance-rejection method and $[5, 11]$ with $\mu = 7.25$, $\sigma = 1.17$ for the spanning-tree method

3.2 Sequential vs. Simultaneous Adding of Edges

Up to this point of study, we have a graph to grow—either one that represents a real search environment or one via the aforementioned random graph-generation method. Except for the brute-force method, no known method guarantees optimal choices in picking the edge(s) to add in order to maximize the algebraic connectivity. The one method we have not incorporated so far is the improved method described in [18], which requires an amount of computational effort less than the brute force method but guarantees to find the optimal solution in case of adding one single edge. If a sequential adding of current-state optimal edges leads to an optimal graph, this method would be our first choice.

We will perform the analysis on a P_{10}, that is, a path with ten nodes and consequently nine edges. Besides the fact that this is a special graph, it shows some important phenomena too. We add a missing edge to the graph, calculate the resulting algebraic connectivity, remove the edge, and pick another one to add. To exhaust all possibilities, we have to do this 36 times. Now, we pick the edge that provides the highest λ_2. There is no way we could do better. To visualize the result we add all 36 edges to the graph and color them according to the calculated algebraic connectivity. The result is shown in Figure 3.2. In this case, there is no unique optimal solution. We could add $e_{1,10}$ or $e_{2,9}$ and would increase λ_2 to the same optimal value.

Let us add $e_{1,10}$ and perform the same calculation for the remaining 35 missing edges. Figure 3.3 illustrates the outcome. This highlights another phenomenon. No matter which edge we add in this state, the algebraic connectivity will stay the same. We have to add at least two edges to get an improvement in λ_2. Pictorially speaking, we created an cycle (C_{10}) by adding $e_{1,10}$ and thus each node is as good as each other. We can rotate the graph and cannot tell any difference. This also proves that λ_2 is non-decreasing instead of increasing with the addition of edges.

If we do our calculation for more than one edge simultaneously, we have to conduct $\binom{36}{2} = 630$ calculations to find the optimal solution for adding two edges (Figure 3.4) and $\binom{36}{3} = 7140$ calculations for adding three edges (Figure 3.5). We observe that the first added edge, $e_{1,10}$, is not part of the solution for two simultaneously added edges. On the other hand, the solution for three simultaneously added edges contains the solution for two simultaneously added edges. Starting from Figure 3.4 and adding the best edge in order to maximize λ_2 will result in Figure 3.5.

This simple example shows two points. First, conducting brute force for more than one edge to add is not practicable, and second, sequentially adding the current graph's best edge will not necessarily return the graph with the highest possible algebraic connectivity. Therefore, a heuristic method to find the next edge, although this may not be the best one, is good enough to grow the graph sequentially. For that reason, we prefer the computationally easier Fiedler vector method to the in [18] proposed algorithm.

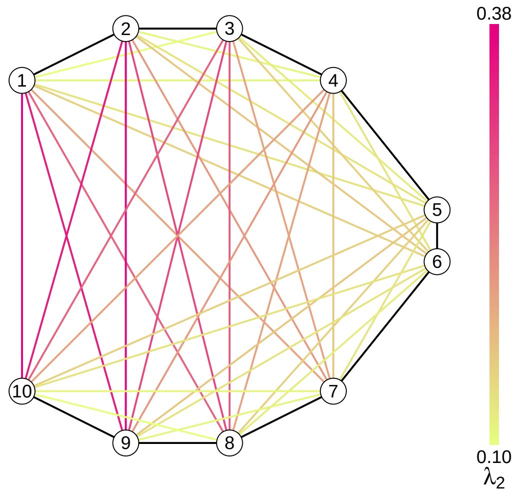


Figure 3.2: **Ten-node graph, sequential adding of first edge** - Adding one of the colored edges will increase λ_2 according to the edge color.

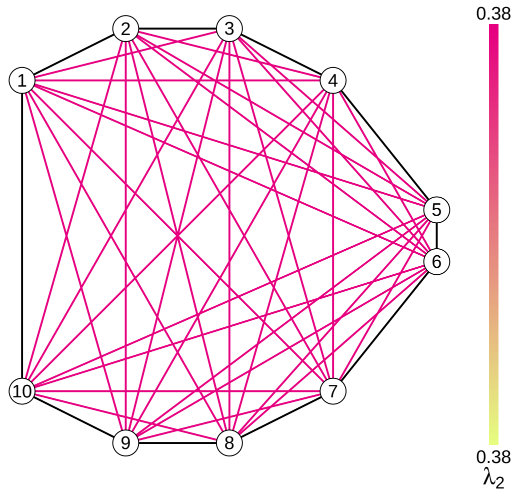


Figure 3.3: **Ten-node graph, sequential adding of second edge** - No matter which edge will be added, λ_2 will keep its value.

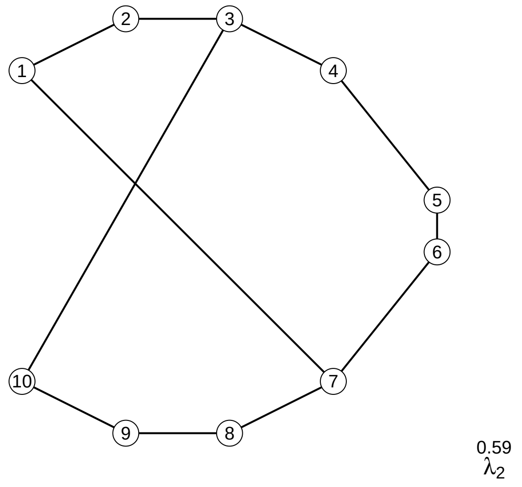


Figure 3.4: **Ten-node graph, simultaneous adding of two edges** - This is the optimal solution for adding two edges with optimal value for λ_2. This neither includes the edge $e_{1,10}$ nor $e_{2,9}$ witch are the optimal solutions for adding one edge.

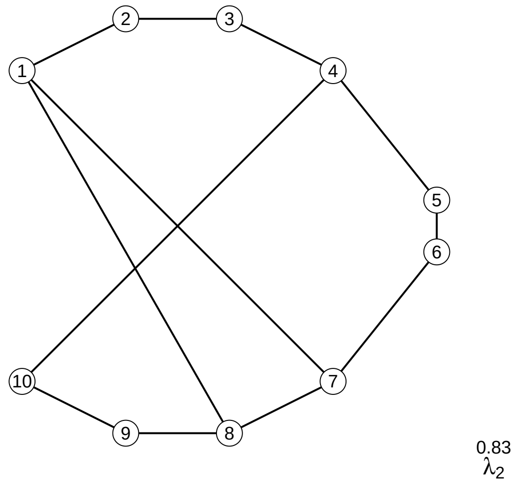


Figure 3.5: **Ten-node graph, simultaneous adding of three edges** - This is the optimal solution for adding three edges with optimal value for λ_2. This includes the optimal solution for adding two edges.

3.3 Fiedler Vector vs. SDP

So far, we have the choice between the sequential Fiedler vector method and the semidefinite program (SDP) formulation [16] to grow a graph according to its algebraic connectivity. Using the SDP, we have to conduct one calculation to find k edges to add at once. Remember that the SDP only works if we relax the problem and allow solutions with fractions of edges. The optimal value is the upper bound for λ_2. At the end, we have to round the optimal solution to get a heuristic integer solution. The heuristic solution will have an optimal value less than or equal to the upper bound. Another problem could be the SDP formulation itself. We have to store the $n \times n$ matrix $a_e a_e^T$ for each of the $(\frac{n(n-1)}{2} - m)$ nonexistent edges in $\mathcal{G}(\mathcal{V}, \mathcal{E})$, with $|\mathcal{V}| = n$ and $|\mathcal{E}| = m$. This will use a lot of memory, especially for sparse graphs, and the computational effort hits the limit pretty soon. We can investigate whether the gain is worth the effort.

We can generate a connected random graph with 20 nodes and constrain the number of edges to be between 20 and 30 in the first step. For each of the missing edges we can calculate λ_2 (that is, the algebraic connectivity) after adding that single edge to the graph. The edge with the highest value is the optimal solution we are looking for. This time, we keep the best ten edges in mind, just for comparison. Now we look at the SDP integer solution for the same graph and match it to the ten-best known edges, if possible. We do the same for the Fiedler vector method and repeat this process for 2000 different random graphs. The plot for graphs with different edge densities is shown in Figure 3.6.

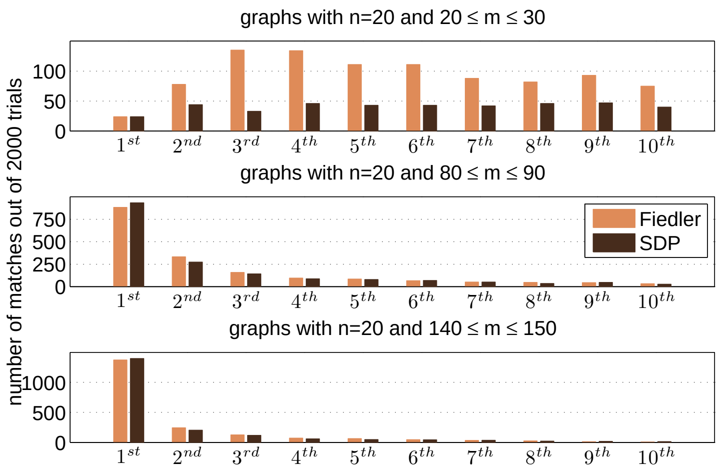


Figure 3.6: **Fiedler vs. SDP, add one edge to a random graph** - The edges found by the Fiedler vector method and by a semidefinite program were matched to the ten best known edges (calculated by brute force). The plot shows the number of times a method hit a specific edge. The simulation was repeated 2000 times for each different edge density with $n = 20$.

Notice that the expected number of missing edges is $\frac{20 \cdot 19}{2} = 165$, if we constrain $20 \leq m \leq 30$.

Thus, the number of matches for the best (second best, third best ...) edge would be $\frac{2000}{165} = 12.12$ if we just randomly pick an edge. So both methods do much better. How well they do depends on the sparsity of the graph. The smaller the edge set to pick from, the better the result.

Other than the Fiedler-vector method, the SDP can simultaneously find more than one edge to add. We have not incorporated this advantage, so far. Let us do the same simulation and grow the graph by 10, 20, and 40 edges. Unfortunately, we do not know the optimal solution and brute force is just not going to work in reasonable time. For that reason, we switch the measure of performance to the resulting algebraic connectivity. We count, which method returns the graph with higher λ_2. The results are plotted in Figure 3.7. This analysis shows that the greedy

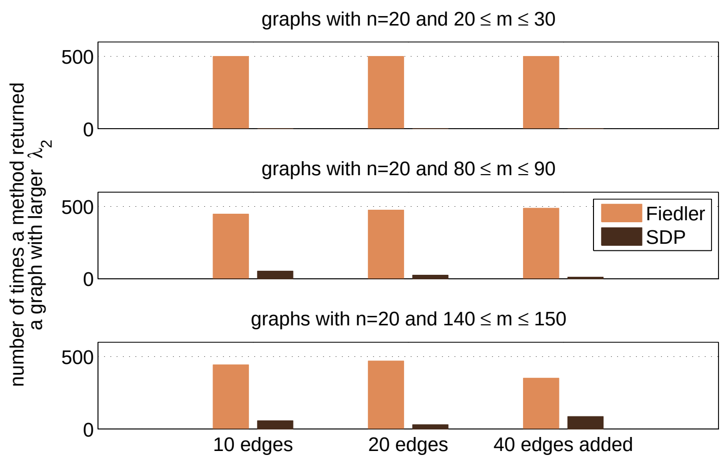


Figure 3.7: **Fiedler vs. SDP, add more than one edge to a random graph** - A random graph was grown by a certain number of edges using the Fiedler vector method and a SDP. The plot shows the number of times a method returned a graph with higher λ_2 than the other method. The simulation was repeated 500 times for each different edge density and each different number of edges to add.

algorithm, that is, the sequential Fiedler vector method, is the best method we have found so far. The algebraic connectivity of the grown graph is not optimal, but better than a SDP result after rounding to integer values. For each edge we want to add to the graph, we have to compute the graph Laplacian. The eigenvalue and eigenvector calculation takes approximately $O(\frac{4}{3}n^3)$ operations. To lower the amount of computational effort, we picked more than one edge (block) after one Fiedler vector calculation. For Figure 3.8, we used a block size of 3 and 10 and compared the result to the normal Fiedler vector method (block size 1) and random addition of edges. The result for a block size of three is not bad at all and thus this is a reasonable approach to lower the computational effort. The bigger the block size, the closer the result to that achieved by randomly adding edges.

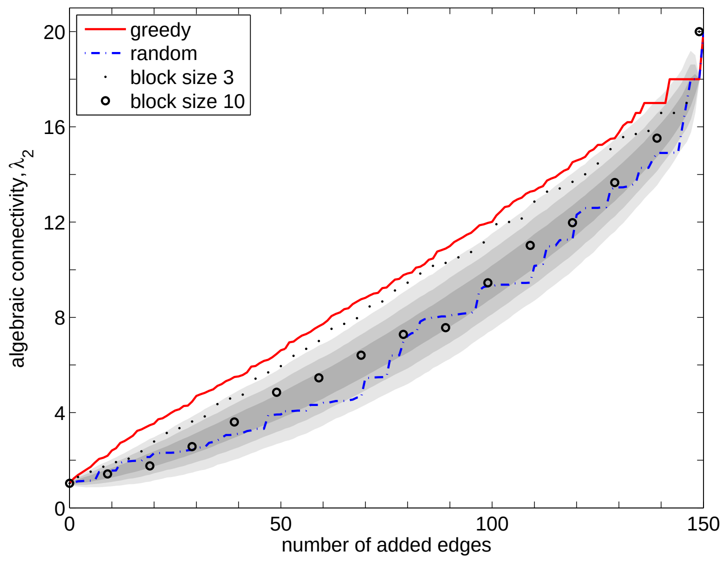


Figure 3.8: **Different methods to grow a graph** - The initial random simple, connected graph with $n = 20$ and $m = 40$ grew to a complete graph using different methods to choose the next edge or block of edges, respectively. The shaded areas represent one, two, and three standard deviations for the randomly grown graph (1000 replications used).

3.4 Relationship between TTD and Algebraic Connectivity

At this point, we have collected all the needed tools to concentrate on the actual objective of this thesis. We are able to generate an arbitrary random connected graph and grow it by a number of edges using the greedy algorithm that utilizes the Fiedler vector.

We are going to generate 100 random graphs with n=50 and conduct searches on each. We assume a perfect sensor (α=β=0) that behaves myopically. We initially place the stationary target in any of the 50 search cells according to a uniform prior distribution. The initial belief is 1, that is, the target is known to be present in the search region. We know the target is inside the search environment but we do not have any information about the likelihood of its location. We always place the searcher in the same start cell (node 1). During the search we record the number of steps the searcher conducts its search until it finds the target. This is equal to the number of cells searched. It will take the searcher at most 50 steps to find the target if the search environment happens to be a complete graph. We repeat the search for each graph 1000 times with different target locations. The time to decision (TTD) is the mean number of steps needed for every run. In the case of our setup, this is the expected number of steps until the searcher finds the target. Figure 3.9 shows once again that λ_2 is non-decreasing with m. The fact that $\lambda_2 \leq \delta$ explains the nearly quadratic shape. Sparse graphs will have small minimum degrees, δ, and it takes many more additional edges to increase δ. Suppose we have a graph

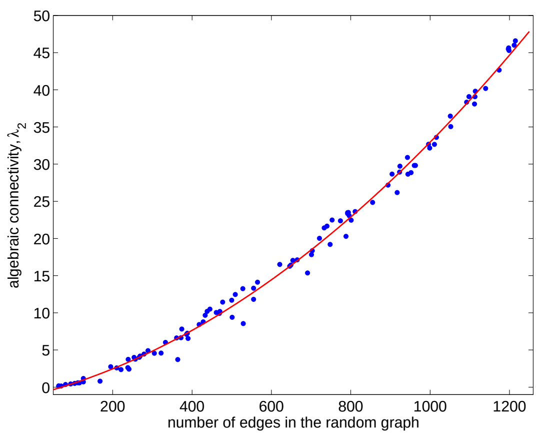


Figure 3.9: **Connectivity in relation to number of edges** - The algebraic connectivity was computed for 100 random, connected graphs with n=50.

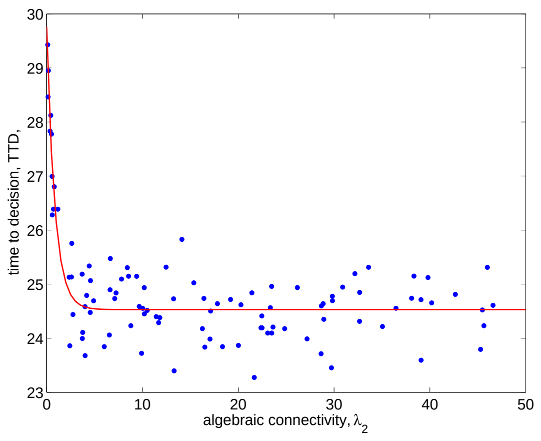


Figure 3.10: **TTD in relation to graph connectivity** - 1000 myopic searches were conducted on each of the 100 graphs. The searcher was considered perfect and the target was located equal likely to any node. TTD is the mean of the 1000 search times for each graph.

with just one missing edge, an almost-complete graph. Adding that one edge will increase λ_2 by at least one. In fact, Fiedler [7] has proven that if $\mathcal{G}$ is a graph with n vertices, which is not complete than $\lambda_2(\mathcal{G}) \leq n - 2$. A corollary to this result is that the last edge needed to complete a graph increases the algebraic connectivity by at least two.

The plotted times to find the target in Figure 3.10 confirm our initial claim that search performance increases with increasing algebraic connectivity. The lower λ_2, the sparser the graph and the longer it takes to visit all the nodes if the searcher behaves myopically. Recall that the searcher moves to the cell with highest probability among all its neighbors. If all the neighbors are already visited and thus have a probability of zero (due to the assumption of a perfect sensor), the searcher does not have any information where to go and can bounce around for some time. This becomes more unlikely the denser the graph is. In a complete graph, the searcher will always have at least one neighbor with allocated probability greater than zero, provided the target is not found yet. But once we have a graph that has a number of edges in the middle part of the plot, there is not much gain in search performance due to adding additional edges.

In addition, we have to consider the variability caused by the target's location and the search behavior. The searcher can find the target in just one step or it can take a very long time and both scenarios are equal likely. Furthermore, it is unlikely for the searcher to take the same route even with the same initial conditions. Whenever there is no clear information for where the searcher should go, it picks randomly from the set of all adjacent nodes.

3.5 Growing Graphs by the Greedy Algorithm

Let us finally grow the first graph by the greedy algorithm. To keep the simulation time short, we start with a sparse graph with 20 nodes. The acceptance-rejection method returns a graph $\mathcal{G}$ with m = 27 edges. We use the greedy algorithm and grow $\mathcal{G}$ 46 times by three edges each. So, $\mathcal{G}$ grows to an almost-complete graph, in steps of three edges. We conduct 2000 searches for each growing state. The searcher is considered perfect $(\alpha = \beta = 0)$ and behaves myopically. The target's location is uniform among the nodes. The plot in Figure 3.11 shows the expected number of steps to find the target, that is, the mean of the 2000 search results for every state of the graph. For comparison, we grow the same graph 46 times by three randomly chosen edges each. We use the second y-axis to display the change of algebraic connectivity. The result is

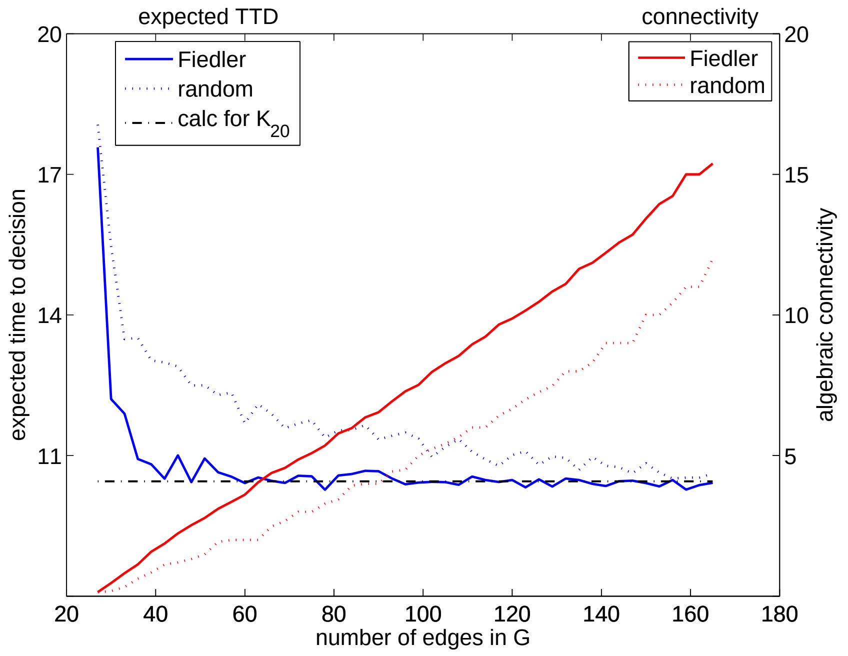


Figure 3.11: **Change of TTD with adding edges by greedy algorithm** - The initial graph with $n = 20$ and $m = 27$ was grown in steps of three edges by the greedy algorithm. 2000 myopic searches were conducted for each growing state. The searcher was considered perfect and the target was located uniformly. The same graph was grown by the same number of edges randomly chosen. Searcher and target properties stayed the same. We included the calculated expected number of steps for a K_{20}.

very interesting. We can improve the search performance by adding just a few wisely chosen edges to the initial graph. Doing this, we are in the neighborhood of the expected time to find the target if the graph were complete. We cannot do better than that. Conversely, adding more edges to that already grown graph does not gain much more.

However, we must be careful with interpretations. We did not constrain the edge set from which to choose the edges and we did not allocate real coordinates to the nodes. So far, it takes the same time (one step) to move from one node to any other adjacent node. These edges are not weighted by real distances. Section 3.6 investigates the effect of different transit times between nodes.

As mentioned, we will find the best search performance on a complete graph. The myopic searcher will visit the nodes of a K_n according to the probability the target is located at that cell. If the searcher is perfect, it will never visit the same cell twice. We use this information to calculate the expected number of cell searches, $E[S(K_n, \frac{1}{n})]$, on a K_n where the target is located in any cell with probability $\frac{1}{n}$, by

$$E[S(K_n, \frac{1}{n})] = \begin{cases} 1 & \text{, with probability } \frac{1}{n} \\ 1 + E[S(K_{n-1}, \frac{1}{n-1})] & \text{, with probability } (1 - \frac{1}{n}). \end{cases}$$

If we know that the target is located in an environment represented by a K_2, we have to search one of the cells to know its location. Setting the initial conditions $E[S(K_2, \frac{1}{2})] = 1$ and solving the recurrence gives the closed form:

$$E[S(K_n, \frac{1}{n})] = \frac{\sum_{k=2}^{n} k}{n} = \frac{n\,(n+1) - 2}{2n}, \tag{3.1}$$

$$Var[S(K_n, \frac{1}{n})] = \frac{1}{n}\left(\sum_{k=1}^{n-1}\left[\left(k - E[S(K_n, \frac{1}{n})]\right)^2\right] + \left((n-1) - E[S(K_n, \frac{1}{n})]\right)^2\right).$$

Now, we have to analyze if we always can improve the search performance in that way rather than as artifact due to the chosen graph. We are going to generate fifteen sparse graphs with $n = 20$ and grow those graphs ten times by three edges each step. We conduct 2000 myopic searches with perfect sensor for each of the growing states. The target is still located equal likely at one of the search cells. The result is shown in Figure 3.12.

We repeat this simulation for sparse graphs with 50 and 100 nodes. Because we expect a higher number of edges needed to add to increase search performance, we will add five and ten edges, respectively, at every step. The results are shown in Figure 3.13 and Figure 3.14.

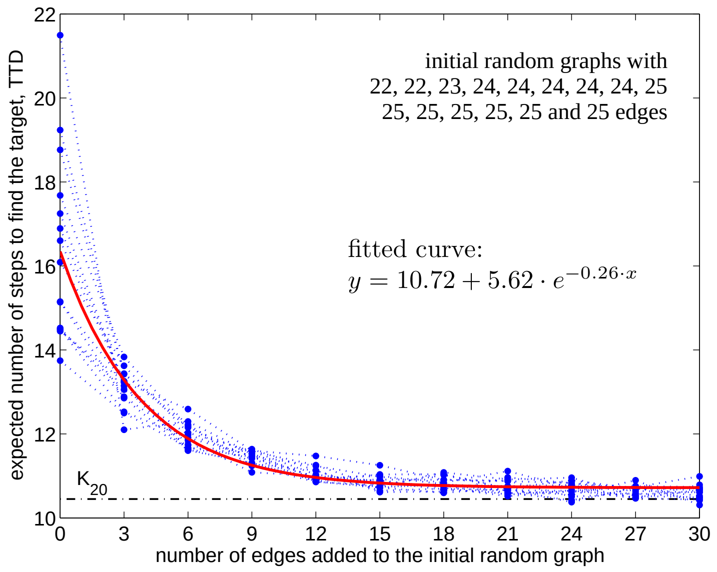


Figure 3.12: **Conducting searches on fifteen growing sparse graphs with 20 nodes** - The initial graphs with $n = 20$ and $20 \leq m \leq 25$ were grown in steps of three edges by the greedy algorithm. 2000 myopic searches were conducted for each growing state. The searcher was considered perfect and the target was located uniformly. We included the calculated expected number of steps for a K_{20}.

For the plots in Figure 3.12–3.14, we can fit an exponential function of the form

$$a + b \cdot e^{c \cdot x},$$

where x is the number of added edges to the initial sparse graph. Using this fitted curve equation, we can estimate the number of edges one must add to a sparse graph in order to increase search performance by a desired amount. The denser the initial graph, the more edges are needed to get a significant decrease in expected time to find the target, demonstrating the effect of diminishing returns by additional edges. The actual number of edges to add is seen to be dependent on the number of nodes in the graph. However, common among all graphs and independent of the number of nodes is the fact that at some point, there is no further improvement with the addition of more edges.

To illustrate the value of the fitted model, consider that for a sparse graph with $n = 50$ (refer to Figure 3.13), one can add just ten edges to decrease the search time by half relative to the lowest possible time, i.e., the theoretically derived lower bound. In other words, we gain 50% of the possible search improvement by adding just 0.857% of the total possible edges available for addition. The analytic formulation that captures this relationship is one of the main contributions of this work.

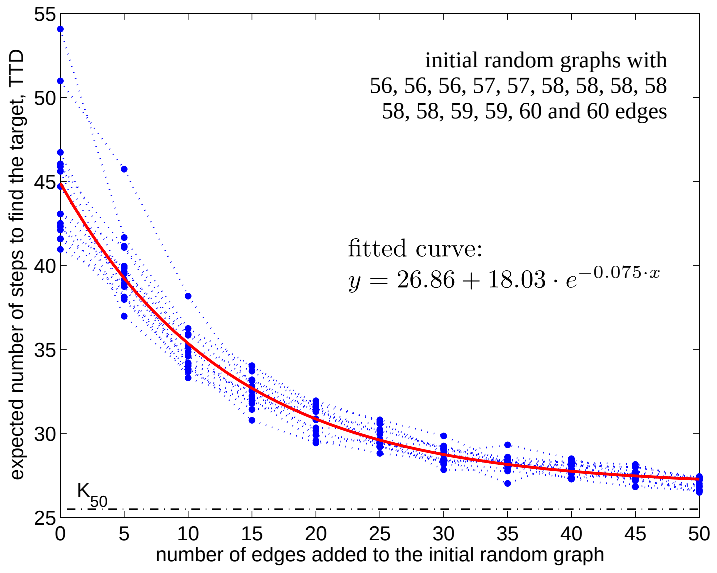


Figure 3.13: **Conducting searches on fifteen growing sparse graphs with 50 nodes** - The initial graphs with $n = 50$ and $55 \leq m \leq 60$ were grown in steps of five edges by the greedy algorithm. 2000 myopic searches were conducted for each growing state. The searcher was considered perfect and the target was located uniformly. We included the calculated expected number of steps for a K_{50}.

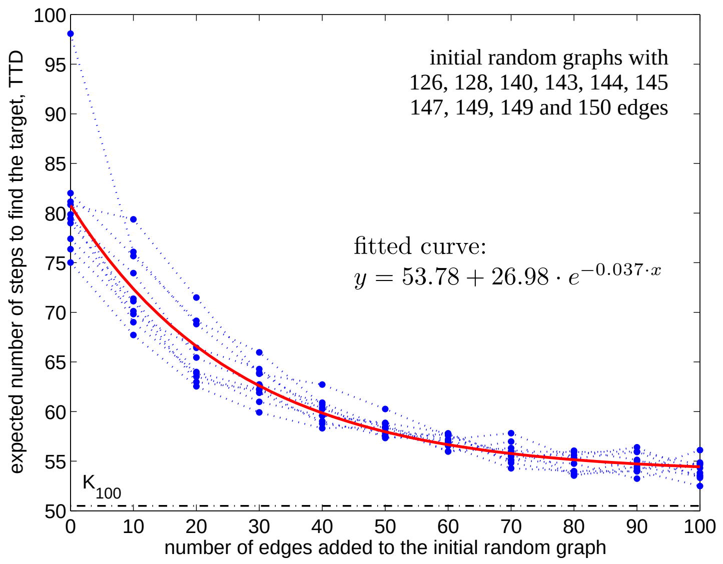


Figure 3.14: **Conducting searches on ten growing sparse graphs with 100 nodes** - The initial graph with $n = 100$ and $120 \leq m \leq 150$ were grown in steps of ten edges by the greedy algorithm. 2000 myopic searches were conducted for each growing state. The searcher was considered perfect and the target was located uniformly. We included the calculated expected number of steps for a K_{100}.

The way we came up with the equation to explain the relationship between added edges and time to decision is time consuming. We cannot provide an equation for a graph with an arbitrary number of nodes unless we conduct the simulation. Thus, it would be nice if we could estimate the parameters a, b and c in relation to n, the number of nodes in the graph.

We space a series of simulations with different numbers of nodes. For each n, we take a number of graphs, grow those graphs, and fit an exponential line. To get a good estimate for the parameters, we have to grow the graphs until it is complete. Remember that the number of edges is dramatically increasing with the number of nodes. Knowing that the fitted line is exponential, we can increase the number of edges between the data points, i.e., starting with three edges between two simulations, increase to five, ten, 100, and 500 for a large number of nodes. Another factor that influences the accuracy of the parameter estimates is the number of searches conducted. While 2000 replications for a graph with $n = 10$ is sufficient, the same number of replications for a graph with $n = 120$ is questionable. The target is uniformly distributed among the nodes. The long-run number of times the target is located in cell c_i is 200 for $n = 10$ and 16.67 for $n = 120$.

In Figure 3.15 we can see the simulations result for the parameter a. The value turns out to be

$$a = E[S(K_n, \frac{1}{n})].$$

In Figure 3.16 we see the simulation result for parameter b. The value turns out to be

$$b = \sqrt{Var[S(K_n, \frac{1}{n})]}.$$

Figure 3.17 shows the simulation result for parameter c. We included an exponential fit. A close approximation for the value of c is

$$c = \frac{n}{\sqrt{Var[S(K_n, \frac{1}{n})]}} \cdot (E[S(K_n, \frac{1}{n})] \frac{1}{\frac{n(n-1)}{2} - n}.$$

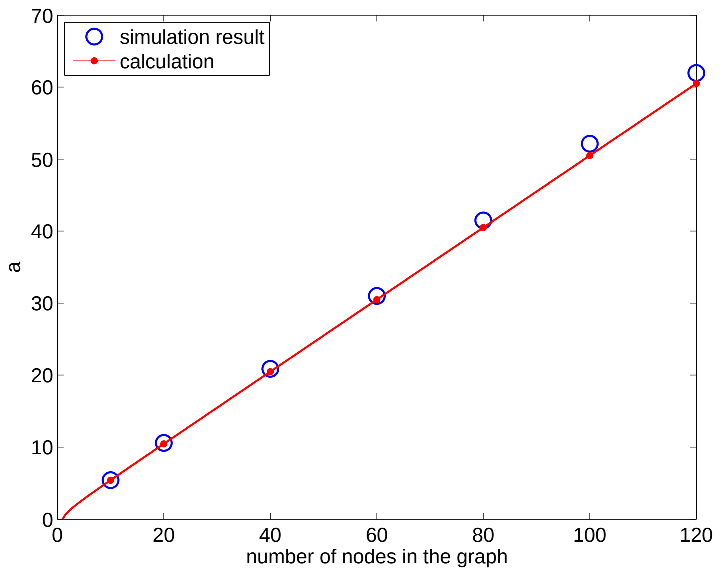


Figure 3.15: **Estimate coefficient $\mathbf{a}$** - Simulations to grow graphs with different n were conducted. The plot shows the parameters a of the fitted curve. The calculated numbers are $a = E[S(K_n, \frac{1}{n})]$.

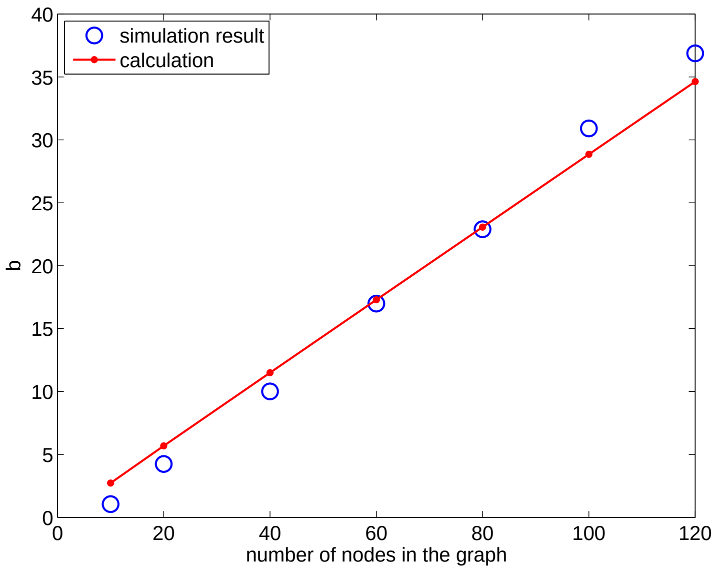


Figure 3.16: **Estimate coefficient $\mathbf{b}$** - Simulations to grow graphs with different n were conducted. The plot shows the parameters b of the fitted curve. The calculated numbers are $b = \sqrt{Var[S(K_n, \frac{1}{n})]}$.

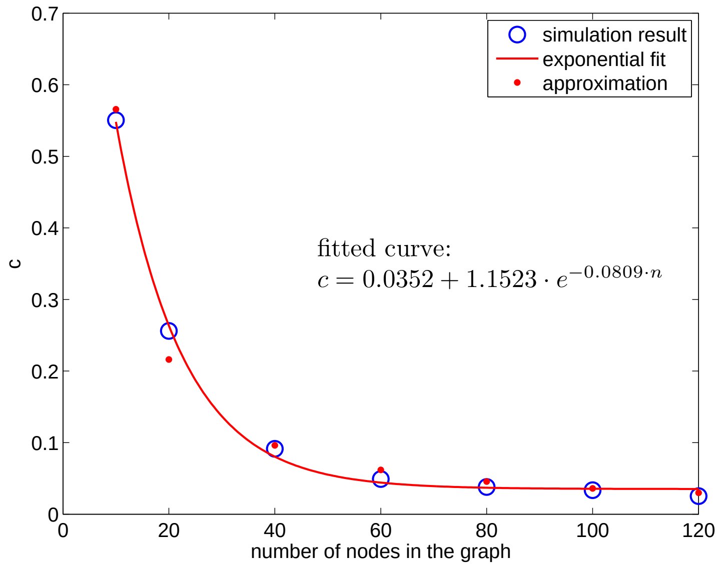


Figure 3.17: **Estimate coefficient $\mathbf{c}$** - Simulations to grow graphs with different n were conducted. The plot shows the parameters c of the fitted curve. We includes an exponential fit as well as our stated approximation.

3.6 Incorporating Initial Information

Up to this point, we only used random graphs without any coordinates assigned to their nodes; that is, we assumed the length of each edge to be one. In real applications, however, we certainly have to think about the edge length and the associated travel time. In addition, we may have some information about the likelihood of the target location, described by the prior probability map. Using that probability map to determine the path taken through the graph can decrease the expected time to find the target. In this section we describe the impact of including this information into the search process.

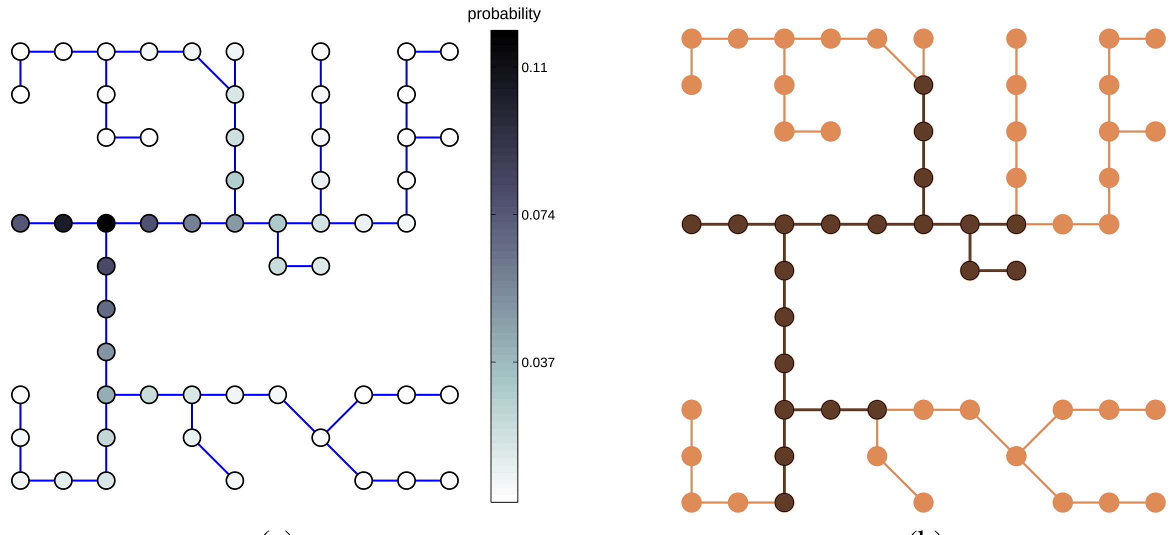
3.6.1 The Subgraph Idea

The simulations and results so far are based on a prior probability map that is uniform, i.e., the target could be located anywhere. Fortunately, this is not true all the time, since prior intelligence is often available. If we provide a probability map to the searcher that is not uniform, the target could be found faster. The improvement in TTD depends on the distribution itself. It is more likely to find the target in one of the high-probability cells. Clearly, the best tactic is to search the cells in order of their assigned probability from highest to smallest—see "most inviting strategy" [54]. Exactly that will happen if the search environment is a complete graph. We have already made use of this fact when calculating the minimum time it takes to find the target. Unfortunately, a complete graph is hard to find in real-world applications and therefore we will not be able to reach the desired cell within the next step.

Here is another approach. Once the searcher reaches the area of high-probability cells, we want it to visit as many of those cells as possible before it searches low-probability cells. An induced complete subgraph that includes only high-probability cells would do the job. After entering the subgraph, the searcher is focused on all the nodes with a higher likelihood of finding the target. It sounds beneficial to add edges in order to improve the connectivity among the higher probability nodes.

Consider the implementation as follows: We extract a subgraph from the search environments model that consists of all the high-probability cells, such that the probability sums up to a predefined threshold (e.g., 80%) of the probability mass allocated to the search environment. Recall that the probability inside the search environment does not have to sum to one because of the introduction of the virtual cell. Figure 3.18 depicts this idea and we can see the original graph, the associated probabilities map, and the associated induced subgraph.

After finding the subgraph, we can now extract it from the original graph. That subgraph is subjected to growth by the greedy algorithm described in previous sections and the augmented subgraph is merged back into the original graph. Using this method we ensure that the growth algorithm will only consider the high-probability nodes when choosing to add an edge. The result should provide more possibilities to the searcher to walk around the high-probability cells.

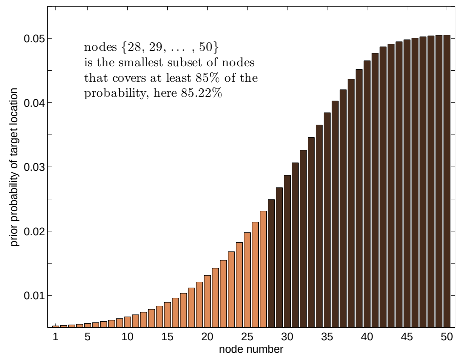


(a) (b)

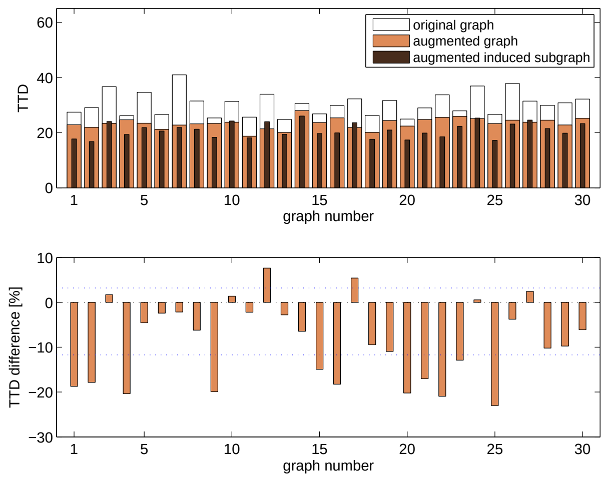
Figure 3.18: **Induced subgraph** - **(a)** The probability map was derived using a Markov chain, the same way as described in Figure 3.20. **(b)** The darker nodes of the induced subgraph represent the smallest node set needed to represent 80% of the probability mass. That subgraph is subject to grow by the greedy algorithm.

We created 30 sparse random graphs with 50 nodes and number of edges between 60 and 70. We assigned a prior probability map that looks Gaussian on the graph. Each of the graphs was grown by ten edges using the greedy algorithm with and without extracting a subgraph that incorporates 85% of the probability. Finally we conducted 3000 searches on each of the 90 graphs to assess the goodness of the subgraph idea. Figure 3.19 shows the probability distribution and the simulation result.

We can observe that the subgraph method outperforms the ordinary greedy algorithm in 80% of the cases. In other cases the subgraph method did worse. The amount we could gain using this subgraph approach is much more than the amount we would lose. Therefore, there is a potential value in using this method, which merits further investigation on what conditions we must meet in order to see an improvement.



(a)



(b)

Figure 3.19: **Subgraph analysis, random graphs** - **(a)** Probability map that was used for the analysis. **(b)** Time-to-decision comparison for the different methods. The plot at the bottom shows the difference gained in TTD using the subgraph idea. Reference is the improvement gained by adding ten edges to the original graph using the greedy algorithm. The horizontal lines mark the averages of gain or loss.

3.6.2 Coordinates and a More Realistic Edge Length

The assumption of having equal edge length makes the calculation easy and there is no need to assign actual coordinates to the nodes. Certainly, that is not true dealing with real search environments. Let us fix that and see how it influences the results we have observed so far.

A real scenario will be given by an environment that has no or just a few search nodes on it, e.g., a city map. We can still keep the equal-edge-length assumption by discretizing the search environment in an appropriate way. In other words, we put the nodes to the graph such that all the edges have the same length or at least fall into a range of edge length. In addition, we will leave the area of totally random graphs and will use graphs that could have an interpretation in the real world (e.g., a road network, the floor plan of a building). Figure 3.20 depicts such a graph. We take an arbitrary road network, put a grid on top, and assign nodes such that no edge length exceeds $\sqrt{2}$.

If there is any information about the target's location within that search environment, we can express it as a prior probability of the target's presence in each of the search cells (cell belief). The assigned prior probabilities should represent a possible real world scenario. Thus, randomly assigning probabilities seems to be a bad idea. Suppose we model a part of a street by ten nodes that build a simple path. We would like not see an alternating sequence of very high and very low probabilities following that path in a real world problem.

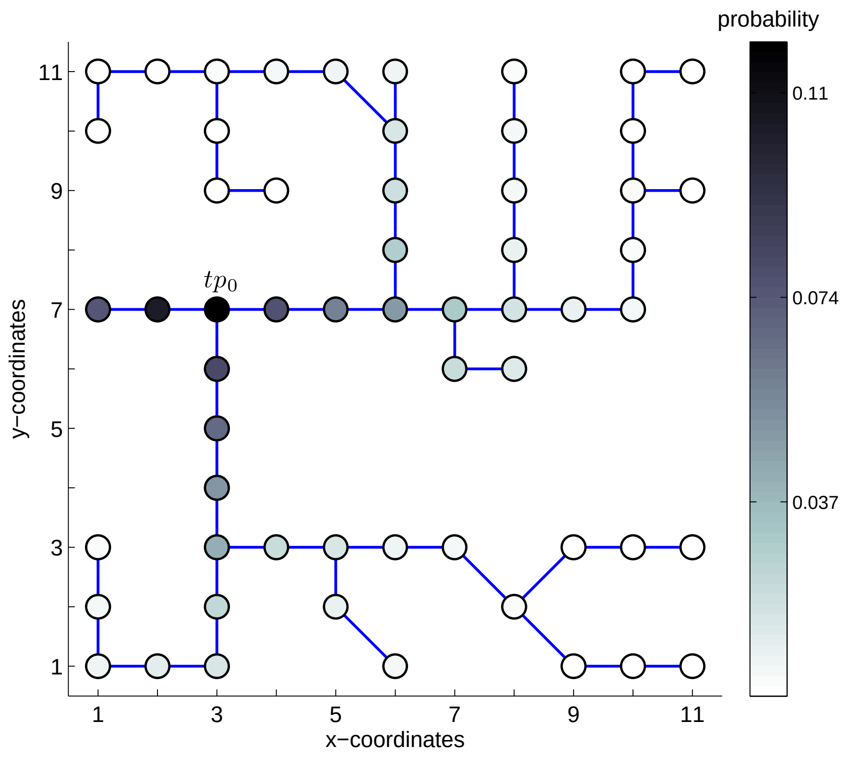


Figure 3.20: **Graph with assigned coordinates and prior target distribution** - The target was assumed to be at position tp_0 at time zero with probability one. The likelihood of the targets position after 40 time steps is coded by the assigned colors. The calculation was done using a Markov chain.

A Markov chain [55] can be used to calculate reasonable probabilities to model a target's movement. Assuming the same probability that the target will stay in the current node or move to any of the adjacent nodes, we can easily derive the transition matrix, $\mathbf{P}$, from the adjacency matrix. The $1 \times n$ start vector, $\mathbf{sv}$, allows us to distribute the initial probability mass among more than one search cell. Knowing the target's position probabilities at time $t_s = 0$, we can calculate the probability map, denoted ***probMap***, that represents the target's location likelihood after t_s time steps:

$$probMap = \mathbf{sv} \cdot \mathbf{P}^{t_s}.$$

In Figure 3.20, we can see such a prior probability coded by different colors. The target was assumed to be at the position marked by tp_0 at time zero. The searcher cannot reach the search area within the next 40 time steps. By the time the searcher arrives, the target could be in each of the search cells with a probability coded by the cell's color, incorporating the target's maximal speed. Now we can use that information and provide it to the searcher. Compared to a uniform prior target distribution, the searcher's myopic behavior becomes more efficient. We can use the greedy algorithm to grow any graph and certainly we can grow a graph as described above.

In order to keep the ability of transferring the problem to the real world, we have to think about the set of possible edges to add; in particular, we want to constrain that edge set according to edge length.

The greedy algorithm itself does not need any node coordinates to come up with a solution and therefore does not care about the length of the edges it adds. But we do not want to tolerate adding very large edges, where large can be measured by the largest edge already in the graph. Suppose we have a graph that represents an entire city and its road network. We have put effort to ensure that all the edges in the search region model are bounded above. In order to maximize the algebraic connectivity of that graph, the algorithm may want to add an edge that connects the southernmost with the northernmost cell, straight through the city, which is hard to imagine would happen in the real world. Even it the searcher were a UAV, it would take a lot of precious time to travel that long edge without searching.

We can take the graph and the probability map from Figure 3.20 and do various kinds of perturbations. The edges in the graph have a length of 1 or $\sqrt{2}$. We allow the greedy algorithm to add five edges with a total length less than or equal to four. On one hand, we use the ordinary greedy algorithm, and on the other hand, the subgraph method with a probability threshold of 0.9. Starting the search at position $(4, 7)$ gives a better result for the subgraph method.

Figure 3.21 shows a histogram of time to decision for 5000 replications, conditioned on the target location. We observe that augmenting a graph means decreasing the time to decision for some nodes while increasing the time for some other nodes. The subgraph method does well as long as the target is located inside that subgraph. We can decrease the TTD more than augmenting the entire graph. But every coin has its flip side. In case the target is not located inside the augmented subgraph, the search time may be longer than conducting the search without adding any edges to the graph. The reason is because we try to force the searcher to stay in high-probability areas of the search region. But even if small, there is a probability the target is located in a low-probability cell. Providing more opportunities to walk directly between the high-probability cells will postpone the need to consider low-probability for the myopic searcher. Essential for the proposed method is to find the right subgraph, in particular, to find the right threshold to extract the subgraph. We want to set the threshold in order to gain most improvement in search performance whenever the target is located inside the extracted subgraph. At the same time, we do not want to see the TTD increased for the low-probability target locations.

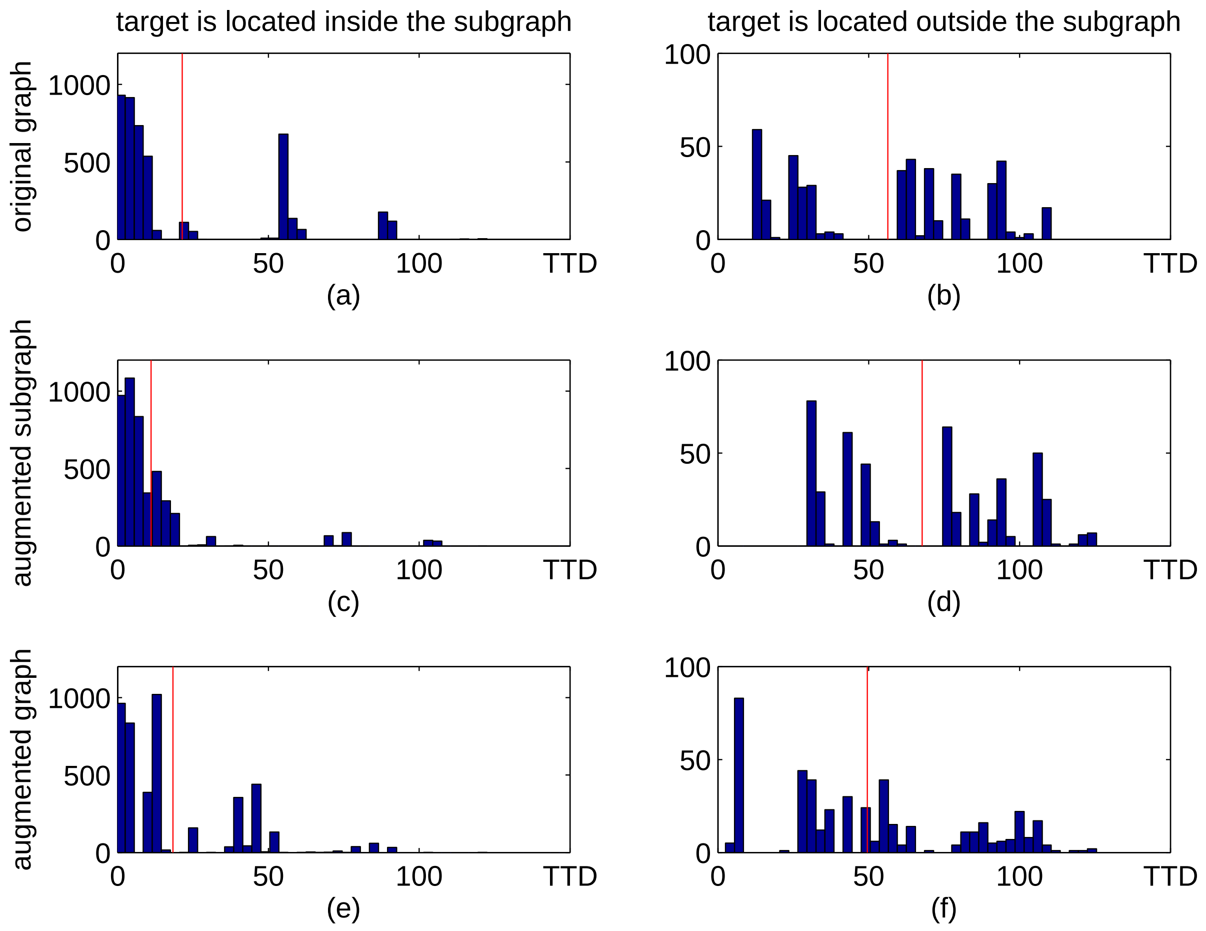


Figure 3.21: **TTD conditioned on target location** - We used the graph and the probability map from Figure 3.20 and conducted a search using the original graph and two different augmented graphs. The searcher was considered perfect. The mean of TTD for each plot is marked by the vertical lines. **(a)** and **(b)** show TTD for the original graph conditioned on the actual target location. Augmenting the graph using a 90% probability subgraph decreased the average TTD whenever the target was located inside the subgraph **(c)** and increased the average TTD otherwise **(d)**. Growing the entire graph decreased the mean of TTD for both conditions by a small amount, as we can see in **(e)** and **(f)**. The overall mean of TTD is smaller, using the subgraph idea for this example.

We can measure the entropy of the probability map to set the threshold used to extract the subgraph [56]. The higher the entropy, the more uncertainty about the target location. Highest uncertainty is reached with a uniform target distribution. At that state, the subgraph idea does not make any sense. The non-monotonicity is explained by Figure 3.21. Adding edges increases TTD for some target locations while decreasing TTD for others. Changing the size of the subgraph results in different added edges, which can cause a change of TTD in the opposite direction.

We have tested this property using the graph and the target distribution from Figure 3.20 for different times late (i.e., number of times the Markov chain was updated for the probability map calculation) and different probability thresholds. The result nominally supports the idea. Let us suppose that we picked the best threshold if we get the lowest TTD. The best threshold increases with increasing entropy of the probability map.

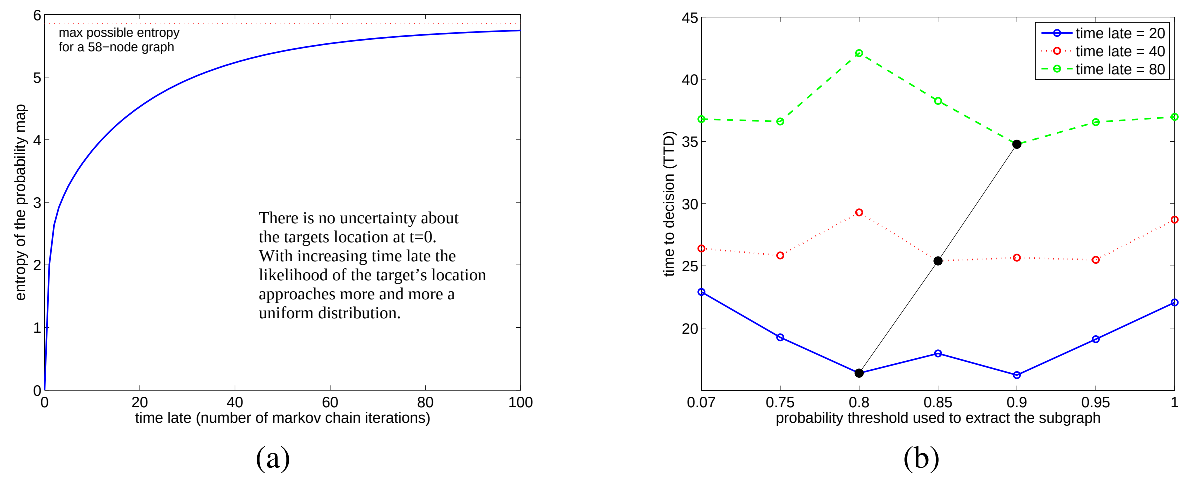


Figure 3.22: **Subgraph analysis, random graphs** - **(a)** Entropy of the probability map according to time late. We used the graph from Figure 3.20 to compute the likelihood of the target location. **(b)** The separate lines show the change in TTD by changing the probability threshold to extract the subgraph. The subgraph is getting larger with increasing threshold. The smallest TTDs for each line are connected. The best probability threshold increases with increasing entropy.

Another part that turns out to be influential is the starting point of the search. To see this, we repeat the simulation used to get Figure 3.21 with several starting points for the search. The simulation result is shown in Table 3.1. The cell with highest probability of hosting the target is cell$(3, 7)$. If we initially place the searcher in cell$(4, 7)$, the right neighbor, we can decrease the search time using the subgraph method rather than augmenting the entire graph. The simulation result for starting the search in cell$(2, 7)$, the left neighbor of the highest probability cell, does not support the use of the subgraph method. We notice that it makes a difference where we place the searcher to start the search. The question about the conditions that make the subgraph method beneficial remains unsolved and opens a possible area for future work.

3.7 Other Search Methods

In this section, we analyze the impact of different search methods. First, we want to know what method best fits more realistic search environments as described in the previous section. Second, we want to test whether our results still hold. In addition, we looked at the assumption of using a perfect sensor.

start node	original graph	augmented subgraph	augmented graph
$(1, 11)$	22.20	22.67	21.65
$(2, 7)$	33.83	16.79	15.42
$(4, 7)$	24.68	16.59	21.20
$(11, 1)$	23.97	24.93	22.72

Table 3.1: **TTD for different start nodes** - Here we see the difference in time to decision caused by a change of the searchers starting position. Cell $(3, 7)$ has got the highest likelihood of target location. While starting from position $(2, 7)$ supports the subgraph method, starting from location $(4, 7)$ requires to grow the entire graph in order to get best search performance.

3.7.1 The Curse of a Myopic Searcher with Perfect Sensor

Although it may intuitively appear beneficial to have a myopic searcher with no false detection rates, it sometimes causes trouble. To investigate the large variance we observed in the simulation data, we looked at the search trace and revealed the problem. Suppose the subgraph in Figure 3.23, with initial probability coded by color intensity. This could be a graph we want to search or just a part of a larger graph that is connected via cell 15 to the remaining nodes. Let us place the target in cell 1 (according to the initial probability there is a high likelihood that this can happen). The perfect searcher will first enter cell 3. The probability allocated to cell 4 is slightly higher than the probability for cell 2. That forces the myopic searcher to search node 4 and all the way down to node 14. All the already searched cells hold the target with probability zero due to the perfect searcher. The memoryless searcher needs to find its way out of that trap without any information. In essence, we find the searcher stuck in a local minimum, which is a known shortcoming of myopic algorithms. At any position (except position 14) the searcher must randomly choose between two equal likely neighbors. So it will walk forward and backwards without pattern until it hits node 3 by chance.

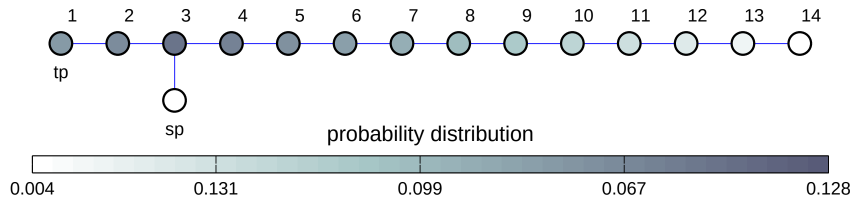


Figure 3.23: **Example for a perfect searchers trap** - The target is assumed to be at position tp. The likelihood of the targets position coded by the assigned colors. The probability allocated to cell 4 is diminutive higher than the probability for cell 2. The perfect myopic behaved sensor will run into a local minimum.

The statistics for 5000 simulations are shown in the following table. For the searcher to avoid being trapped, we would like to give it some memory to retrace the way as soon as possible. We

statistic	min	1^{st}quantile	median	3^{rd}quantile	max	μ	σ
TTD	25	65	103	171	731	133.08	94.73

Table 3.2: **Statistics for a trapped perfect myopic sensor** - Large TTDs are caused by the searcher's lack of memory and the zero probability of target location for the already searched cells.

can use the fact that it is most unlikely to find a perfect sensor in reality and assign a perfect sensor a small false negative rate. That false detection rate causes a missed detection on occasion. It causes the searcher to leave a little probability mass in the already unsuccessfully searched cells. That remaining probability helps the searcher to trace back. It provides information about the way it came and there is no need to randomly choose between two equal likely neighbors. Repeating the same simulation as described above with ($\beta = 0.001$) returned a $TTD = 25$ for each of the 5000 trials, that is, $\mu = 25$ and $\sigma = 0$.

3.7.2 Shortest-path Searcher

We have seen that it takes the least amount of time to find the target if a myopic searcher is placed in a search environment that can be represented by a complete graph with constant edge length. However, we will not see such an environment in real applications. Nevertheless, the takeaway is to search the cells in order of the allocated probability mass, from highest to lowest.

While the myopic searcher uses only local information to identify the next step, a searcher using a constrained shortest-path approach can use global information. It not only considers the adjacent neighbors as possible next search cells, but the entire search environment. It picks the highest probability cell, calculates the shortest path to that cell, and starts to travel.

There are two different behaviors in terms of observations. The searcher could stop at every node while traveling and conduct a search, or the searcher will only search the high-probability node and not perform observations in the nodes it travels through. The second method needs to conduct the same number of searches as the myopic searcher in a complete graph. On top of that time, we have to add the additional travel time. This fact makes it hard to compare against the myopic search method. We need to define how long it takes to travel and how long it takes to search a single cell. In case the searcher moves fast compared to the search time, the shortest-path searcher may have an advantage. If it takes much longer to travel in contrast to search a cell, the myopic searcher may be the better choice.

A simple simulation can show that a shortest-path searcher will travel a lot in a more realistic environment, as shown in Figure 3.20, which is intuitive. The probability decreases from the point of target location at time zero in all possible directions. Searching the nodes in order of decreasing probability forces the searcher to change direction often.

Identifying which method is best for the given search environment and prior probability distribution remains to be determined and is an area for future work. The point we want to make is that our result holds for a shortest-path searcher, too. The time to find the target can be reduced by augmenting the graph.

3.8 Growing Graphs by Hitting Time

Whenever we play the game with imperfect sensors, it may take much longer to find the target compared to the time it takes using a perfect sensor (refer to Figure 2.3). The reason is that we have to search cells multiple times to be sure about the target's presence or absence. A cell where the searcher did not find the target still contains probability mass of the target's presence (provided $\beta > 0$). That probability mass will grow due to future probability updates. To reduce the search time, and more importantly to provide a stopping criterion for the searcher, we previously introduced a lower bound $\underline{B}$ and an upper bound $\overline{B}$, which are the probability thresholds to exceed in order to conclude the search.

As derived in [57], let us suppose we want to concentrate the search on just a single cell until the cell's probability exceeds $\overline{B}$ or gets smaller than $\underline{B}$. We would need the minimum number of searches if the searcher returns only **0** (target not present) or only **1** (target present) respectively.

Utilizing the Bayes' update rule recursively, we can learn the pattern. Provided the sensor returns always zero (one), gives the minimum number t of needed visits

$$t > \left\lceil \frac{\ln \frac{\overline{B}(1-p)}{(1-\overline{B})p}}{\ln \frac{1-\beta}{\alpha}} \right\rceil$$

to exceed the upper threshold $\overline{B}$ and

$$t > \left\lceil \frac{\ln \frac{\underline{B}(1-p)}{(1-\underline{B})p}}{\ln \frac{\beta}{1-\alpha}} \right\rceil$$

to go below the lower threshold $\underline{B}$, where p is the initial probability and α and β the false detection rates. The higher α and β, the more likely it is the searcher has to visit several nodes more than once. The next node with highest probability of containing the target could be far away in distance, and the myopic searcher would spend a lot of time to finally reach that node. The searcher could face the same problem several times. Motivated by this fact, we want to take out the large hitting times from the graph by placing an appropriate edge.

The hitting time (i, j) denoted $ht(i, j)$, is the expected time it takes for the searcher to first visit node j starting at node i [22]. This time can be calculated using an absorbing Markov chain. We declare the probability to stay at node j as one, and thus, created an absorbing state. Rewriting the probability transition matrix in the form $\begin{pmatrix} 1 & 0 \\ R & Q \end{pmatrix}$ allows the calculation $E\left[T|x_0 = i\right] = \sum_{i=s}^{r} (I - Q)^{-1}_{ij}$, where $\{0 \ldots s-1\}$ are the absorbing states and $\{s \ldots r\}$ the transition states. With one calculation, we get the hitting times for node j from all possible start nodes i.

The return time for a node j is the expected time until a searcher revisits that node. We find the solution solving the Markov chain for the long-run proportion of being in a specific state. $E\left[T|x_0 = j\right] = \frac{1}{\pi_j}$. We have to solve the system of equations

$$\pi_j = \sum_k \pi_k P_{kj}$$
$$\sum_j \pi_j = 1.$$

A solution for this is $\pi_j = \frac{deg(j)}{2m}$.

Here is how we can use this information to define a new method to augment a graph. We compute all the hitting times (this is a $n{\times}n$ matrix), find the largest among them, and place an edge between the start and end nodes. Adding an edge using this method influences all the other hitting times immediately. Adding more than one edge to the graph requires multiple hitting-time calculations.

The best method we have so far is the greedy algorithm using the Fiedler vector to select the edge to add. That method is used as a reference to compare the goodness of the hitting-time method. We use a random graph with $n = 50$ and $m = 62$ and grow the graph ten times by five edges. We do this using both the greedy algorithm and the hitting-time method. We conduct a search with 3000 replications for each graph. The searcher has false detection rates of $\alpha = 0.00$ and $\beta = 0.01$. The resulting times to detection are plotted in Figure 3.24.

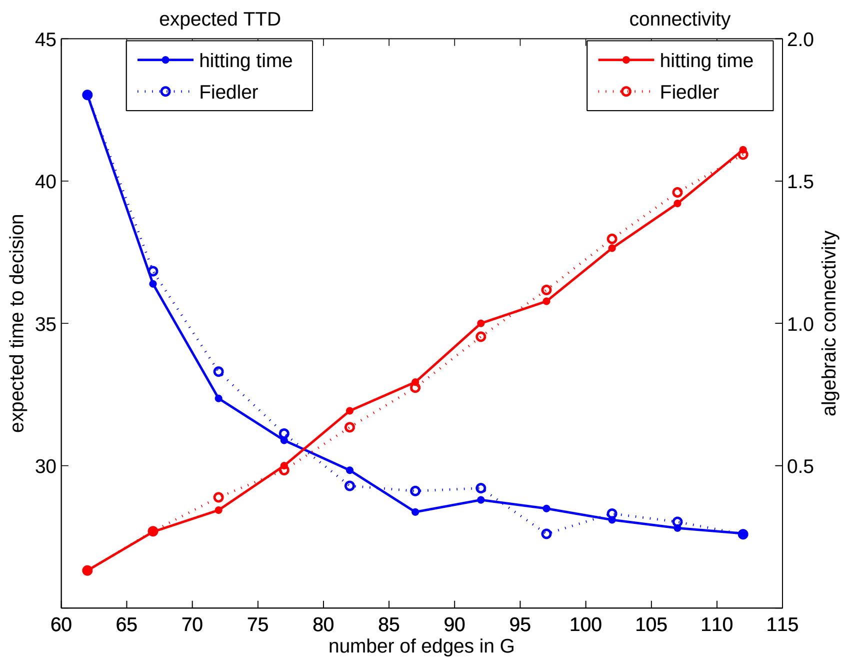


Figure 3.24: **Hitting-time method vs. Fiedler method** - A random graph with $n = 50$ and $m = 62$ was grown ten times to five edges using the different methods. A search with 3000 replications was conducted for every state. The sensor had a false detection rate of $\alpha = 0.00$ and $\beta = 0.01$. In addition to TTD, we plotted the algebraic connectivity for every graph state for comparison purpose.

We observe that the proposed hitting-time method is a good alternative to the Fiedler method.

The hitting time for a specific node is highly influenced by its degree. In case of return time, this fact is obvious looking at the formula for the calculation. The degree difference between $deg(i)$ and $deg(j)$ is one of the reasons for a non-symmetric hitting-time matrix. In most cases, we will see $ht(i, j) \neq ht(j, i)$. Another metric was to use the combined hitting time, that is, using $ht(i, j) + ht(j, i)$, as criterion to find the edge to add. The results for TTD turn out not to be significantly different from the normal hitting-time method, so we can save the additional computation and go for the normal hitting-time method as proposed. Further investigation of this approach is left for future study.

CHAPTER 4: CONCLUSION AND FUTURE WORK

4.1 Conclusion

This thesis investigated the relationship between the addition of edges to a graph representing a search environment and improvement in the search performance, as measured by the time necessary to complete the search. This work leverages existing results pertaining to the second smallest eigenvalue of the graph Laplacian, also known as the algebraic connectivity of the graph. It performs comparisons to identify a greedy method based on the Fiedler vector as an effective, efficient choice for augmenting a graph as well as a method utilizing the maximum hitting time. Statistical simulation studies and analysis using randomly generated graphs validate this finding.

Further, this work enhances the understanding of the positive correlation between λ_2 and the search time until the target is found in problems of probabilistic search. In fact, search performance on sparse graphs can be significantly improved by adding only a few wisely chosen edges, and the relation between number of edges to add and the search time is modeled as exponential. This analytic model offers guidance for the necessary additional number of edges needed to achieve a specified improvement in search for a target. This result also highlights the fact that adding edges beyond a certain number does little to improve search performance, and that knowledge of the relationship between number of additional edges and enhancement of search will limit the expense of adding unhelpful edges.

This work also proposed a novel method for further enhancing the improvement in search performance by partitioning the regions of high and low probability of target presence. By extracting the former region as a subgraph and adding edges using the greedy method found to be most effective, this approach offers a means of incorporating prior information on the target's likely locations into the graph augmentation process.

Simulation studies also show that it is sufficient for the searcher to utilize local information in conducting its adaptive search of the environment. Such a myopic search strategy is seen to outperform other approaches that may involve global information, e.g., shortest path method, but that are wasteful in forcing the searcher to transit through previously visited or low-probability regions in the search environment.

4.2 Future Work

There are numerous avenues for future research, including extensions to account for heterogeneous edge weightings that can represent, for example, physical distances between spatial locations.

Searcher trajectories that address the constrained paths will enhance the relevance of the proposed models to practical applications of probabilistic search. Additional modifications to the model for realistic scenarios will address the need to account for the time it takes to search a cell, in addition to the time it takes to transit there. In such situations, it is possible that a variant of the shortest path search algorithm investigated herein might become more attractive. We note that such constrained search-path optimization problems are computationally intractable [58].

Similarly, more sophisticated treatment of the search-planning process by means of formulating a traveling-salesman problem can further improve the search performance. In other words, slight detours yielding longer search path lengths but accumulating greater probability mass may yield more efficient searches. Such routing algorithms have been formulated in operations research and computer science and can be applied to the probabilistic-search problem as well [59, 60]. A connection between the shortest spanning tree of a graph and the traveling-salesman problem was already claimed in 1956 [61]. That may be a way to further enhance the applied search algorithm. A close cousin of the traveling-salesman problem is the vehicle-routing problem and several known algorithms are discussed in [62]. Especially while using more than one searcher, some of the algorithm's ideas could be adopted to coordinate multiple searchers. Another interesting technique could be the use of pebbles that are left behind while searching, to coordinate the search. That technique has been successfully used searching mazes [63].

The inclusion of correct information pertaining to the target location is seen to be beneficial to the searcher; however, using incorrect information, either by misdirection or by mistake, is significantly detrimental to the search process. In other words, if the searcher erroneously attributes a low likelihood to the true target location, the myopic searcher will search for a long time until that cell is finally searched. At some point it may be worthwhile to stop the search and change strategy. Stopping the search and restarting at a different point is not a bad idea at all [29]. The question is when to stop and where to restart, incorporating the time it takes to reach the restarting location.

However, the myopic searcher may have an advantage over a searcher with global information by restricting the search to local regions. Such advantages may further be accentuated in the case of mobile targets, which is another area of future study [32, 33]. Addition of edges may improve the searcher's ability to localize the target, but also hinders the search by providing greater access for a moving target to maneuver on the graph. Investigation of this trade-off, as well as the ability to block some edges for the target movement, may reveal new methods of search planning. Results from pursuit-evasion problem solving [34, 35, 36, 37, 38] might be considered to guide the searcher through the environment or may reveal other ideas to influence the search environment. While improving the searcher's moving ability, we might want to restrict the target's movement by blocking or deleting edges. The competing strategies are similar to minimax optimization problems. The problem of removing edges that most restrict the target's mobility can be seen as the dual of the thesis' problem. Performing the Fiedler vector method on the graph's complement will find a heuristic solution.

Divide-and-conquer techniques may be beneficial to disconnect the graph in two smaller subgraphs. According to a given budget, we could use two searchers, one for each subgraph, or we could use one searcher, but define where to start searching and when to change to the other subgraph.

Other analytic models relating entropy of the initial target distribution and threshold percentage of probability mass to specify the subgraph to be extracted and augmented offer additional avenues of interesting research.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] B. Bollobas, *Random Graphs*, 2nd ed. Cambridge University Press.

[2] M. E. J. Newman, S. H. Strogatz, and D. J. Watts, “Random graphs with arbitrary degree distribution and their applications,” *Physical Review E*, vol. 64, no. 026118, pp. 1–17, 2001.

[3] M. E. J. Newman, *Random graphs as models for networks*. Wiley – VCH Verlag GmbH & Co. KGaA, Weinheim, 2002.

[4] P. Erdös and A. Rényi, “On random graphs I.” *Publicationes Mathematicae*, no. 6, pp. 290–297, 1959.

[5] ——, “On the evolution of random graphs,” *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, no. 5, pp. 17–61, 1960.

[6] E. N. Gilbert, “Random Graphs,” *The Annals of Mathematical Statistics*, vol. 30, no. 4, pp. 1141–1144, 1959.

[7] M. Fiedler, “Algebraic connectivity of graphs,” *Czechoslovak Mathematical Journal*, vol. 23, no. 98, pp. 298–305, 1973.

[8] ——, “A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory,” *Czechoslovak Mathematical Journal*, vol. 25, no. 4, pp. 619–633, 1975.

[9] B. Mohar, “Eigenvalues, diameter, and mean distance in graphs,” *Graphs and Combinatorics*, vol. 7, no. 1, pp. 53–64, 1991.

[10] N. M. M. d. Abreu, “Old and new results on algebraic connectivity of graphs,” *Linear Algebra and its Applications*, vol. 423, no. 1, pp. 53–73, 2007.

[11] A. S. Ibrahim, K. G. Seddik, and K. J. R. Liu, “Improving Connectivity via Relays Deployment in Wireless Sensor Networks,” in *IEEE Global Telecommunications Conference*. IEEE, Nov. 2007, pp. 1159–1163.

[12] ——, *Connectivity-Aware Network Maintenance via Relays Deployment*. IEEE, Mar. 2008.

[13] Y. Wan, S. Roy, X. Wang, A. Saberi, T. Yang, M. Xue, and B. Malek, "On the Structure of Graph Edge Designs that Optimize the Algebraic Connectivity," in *Proceedings 2008. 47th IEEE Conference on Decision and Control*. IEEE, 2008, pp. 805–810.

[14] D. Mosk-Aoyama, "Maximum algebraic connectivity augmentation is NP-hard," *Operations Research Letters*, vol. 36, no. 6, pp. 677–679, Nov. 2008.

[15] S. P. Boyd, "Convex optimization of graph Laplacian eigenvalues," in *Proceedings of the International Congress of Mathematicians*, vol. 3, no. 1-3. Citeseer, 2006, pp. 1311–1319.

[16] S. P. Boyd and L. Vandenberghe, *Convex optimization*, 7th ed. Cambridge University Press, 2004.

[17] A. Ghosh and S. P. Boyd, "Growing Well-connected Graphs," in *Proceedings of the 45th IEEE Conference on Decision and Control*, ser. (San Diego, CA). IEEE, 2006, pp. 6605–6611.

[18] Y. Kim, "Bisection Algorithm of Increasing Algebraic Connectivity by Adding an Edge," *IEEE Transactions on Automatic Control*, vol. 55, no. 1, pp. 170–174, Jan. 2010.

[19] T. H. Chung, *On Probabilistic Search Decisions under Searcher Motion Constraints*, ser. STAR. Springer Berlin Heidelberg, 2008, vol. 57, pp. 501–516.

[20] T. H. Chung and J. W. Burdick, "A Decision-Making Framework for Control Strategies in Probabilistic Search," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 4386–4393.

[21] G. Brightwell and P. Winkler, "Maximum hitting time for random walks on graphs," *Random Structures & Algorithms*, vol. 1, no. 3, pp. 263–276, Sep. 1990.

[22] L. Lovász, "Random Walks on Graphs: a Survey," Department of Computer Science, Yale University, Tech. Rep., May 1994.

[23] W. K. Grassman, M. I. Taksar, and D. P. Heyman, "Regenerative Analysis and Steady State Distributions for Markov Chains," *Operations Research*, vol. 33, no. 5, pp. 1107–1116, 1985.

[24] H. Chen and F. Zhang, "The expected hitting times for finite Markov chains," *Linear Algebra and its Applications*, vol. 428, pp. 2730–2749, 2008.

[25] F. P. Kelly, “A Remark on Search and Sequencing Problems,” *Mathematics of Operations Research*, vol. 7, no. 1, pp. 154–157, 1982.

[26] F. R. K. Chung, R. L. Graham, and M. Saks, “Dynamic search in Graphs,” *Discrete Algorithms and Complexity*, pp. 351–387, 1987.

[27] P. Sarkar, A. W. Moore, and A. Prakash, “Fast Incremental Proximity Search in Large Graphs,” in *Proceedings of the 25. International Conference on Machine Learning*, 2008, pp. 896–903.

[28] M. Bouvel, V. Grebinski, and G. Kucherov, “Combinatorial Search on Graphs Motivated by Bioinformatics Applications: A Brief Survey,” *Graph-Theory, Concepts in Computer Science*, pp. 16–27, 2005.

[29] T. Walsh, “Search in a Small World,” in *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, vol. 2. Morgan Kaufmann Publishers Inc., 1999.

[30] M. L. Weitzman, “Optimal Search for the Best Alternative,” *Econometrica*, vol. 47, no. 3, pp. 641–654, 1979.

[31] D. Assaf and S. Zamir, “Continuous and discrete search for one of many objects,” *Operations Research Letters*, vol. 6, no. 5, pp. 205–209, 1987.

[32] S. S. Brown, “Optimal Search for a Moving Target in Discrete Time and Space,” *Operations Research*, vol. 28, no. 6, pp. 1275–1289, 1980.

[33] A. R. Washburn, “Search for a Moving Target: The FAB Algorithm,” *Operations Research*, vol. 31, no. 1, pp. 739–751, 1983.

[34] T. Parsons, *Pursuit-evasion in a graph*, ser. Lecture Notes in Mathematics. Springer-Verlag Berlin Heidelberg, 1978, vol. 642, pp. 426–441.

[35] M. A. M. Vieira, R. Govindan, and G. S. Sukhatme, “Optimal policy in discrete pursuit-evation games,” Department of Computer Science, University of Southern California, Tech. Rep. 08–900, 2008.

[36] R. Vidal, O. Shakernia, H. J. Kim, D. H. Shim, and S. Sastry, “Probabilistic Pursuit-Evation Games: Theory, Implementation, and Experimental Evaluation,” *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 662–669, 2002.

[37] C. V. Ravishankar and S. Kopparty, “A framework for pursuit evation games in R^n,” *Information Processing Letters*, vol. 96, no. 3, pp. 114–122, 2005.

[38] A. S. Goldstein and E. M. Reingold, “The complexity of pursuit on a graph,” *Theoretical Computer Science*, vol. 143, no. 1, pp. 93–112, 1995.

[39] D. Assaf and S. Zamir, “Optimal Sequential Search: A Bayesian Approach,” *The Annals of Statistics*, vol. 13, no. 3, pp. 1213–1221, 1985.

[40] F. Bourgault, T. Furukawa, and H. F. Durrant-Whyte, “Optimal Search for a Lost Target in a Bayesian World,” *Field and Service Robotics*, 2006.

[41] A. R. Washburn, “Branch and Bound Methods for Search Problems,” Department of Operations Research, Naval Postgraduate School, Monterey, California, Tech. Rep. 95–003, 1995.

[42] J. L. Devore, *Probability and statistics for engineering and the sciences, enhanced review edition*. Duxbury Press, 2008.

[43] S. M. Ross, “A Problem in Optimal Search and Stop,” *Operations Research*, vol. 17, no. 6, pp. 984–992, 1969.

[44] C. C. J. Milton, “Optimal Stopping in a Discrete Search Problem,” *Operations Research*, vol. 21, no. 3, pp. 741–747, 1973.

[45] D. H. Wagner, W. C. Mylander, and T. J. Sanders, *Naval Operations Analysis*, 3rd ed. Naval Institute Press, Annapolis, Maryland, 1999.

[46] A. R. Washburn, *Search and detection (Topics in operations research)*, 4th ed. Institute for Operations Research and the Management Sciences, 2002.

[47] L. D. Stone, *Theory of optimal search)*, 2nd ed. Military Applications Section, Operations Research Society of America, 1989.

[48] G. Chartrand and P. Zhang, *Introduction to Graph Theory (reprint) (Walter Rudin Student Series in Advanced Mathematics)*. McGraw-Hill Science/Engineering/Math, 2004.

[49] R. L. Rardin, *Optimization in Operations Research*. Prentice Hall International, 1998.

[50] D. P. Bertsekas, *Nonlinear Programming*, S. M. Robinson, Ed. Athena Scientific, 1999, vol. 2.

[51] R. A. Horn and C. R. Johnson, *Matrix Analysis*, ser. Graduate Texts in Mathematics. Cambridge University Press, 1990.

[52] S. J. Leon, *Linear Algebra with Applications*, 7th ed. Prentice Hall, 2006.

[53] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, ser. Texts in Applied Mathematics. SIAM, 1997, vol. 55, no. Section 5.

[54] F. H. Smith and G. Kimeldorf, "Discrete sequential search for one of many objects," *The Annals of Statistics*, vol. 3, no. 4, pp. 906–915, 1975.

[55] S. M. Ross, *Introduction to Probability Models*, 9th ed. Academic Press, 2006.

[56] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. John Viley & Sons, Inc., New York, USA, 1991.

[57] T. H. Chung and J. W. Byrdick, "Analysis of Search Decision Making using Probabilistic Sarch Strategies," in *IEEE Transactions on Robotics*. IEEE, to appear.

[58] K. E. Trummel and J. R. Weisinger, "The Complexity of the Optimal Searcher Path Problem," *Operations Research*, vol. 34, no. 2, pp. 324–327, 1986.

[59] S. Lin and B. Kernighan, "An effective Heuristic Algorithm for the Traveling Salesman Problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1971.

[60] D. L. Applegate, R. E. Bixby, V. Chvatál, and W. J. Cook, *The Traveling Salesman Problem*. Princeton University Press, 2007.

[61] J. Kruskal, Joseph B., "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956.

[62] G. Laporte, "The Vehicle Routing Problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, no. 59, pp. 345–358, 1992.

[63] M. Blum and D. Kozen, "On the power of the compass (or, Why Mazes are Easier to Search than Graphs)," in *Foundation of Computer Science, Annual Symposium*, 1978, pp. 132–142.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A: SOFTWARE USED FOR SIMULATION AND ANALYSIS

Simulation environment

I used the following program for simulation and analysis:

MATLAB® & SIMULINK®
student version
version 7.10.0.499 (R2010a)
32-bit (win32)
The MathWorks™

Besides the help files that come with the program I frequently used the online help that can be found under:

`http://www.mathworks.com/academia/student_version/start.html`

SDP-solver

To be able to solve semidefinite programs within the MATLAB® environment I downloaded and installed the following solver:

cvx: A system for disciplined convex programming package
©2005-2010 Michael C. Grant and Stephen P. Boyd
`http://www.stanford.edu/~boyd/cvx`

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Directory, Training and Education, MCCDC, Code C46
   Quantico, Virginia

   Officer students in the Operations Research Program are also required to show:

4. Director, Studies and Analysis Division, MCCDC, Code C45
   Quantico, Virginia

   Officer students in the Space Ops/Space Engineering Program or in the Information Warfare/Information Systems and Operations are also required to show:

5. Head, Information Operations and Space Integration Branch,
   PLI/PP&O/HQMC, Washington, DC